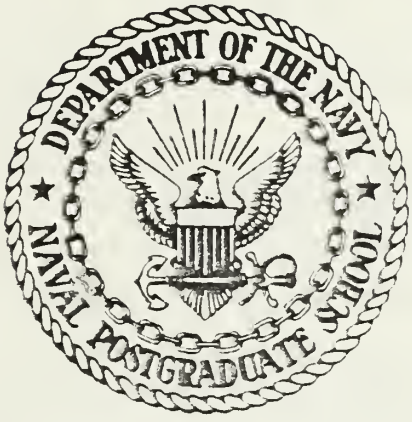
1986

An ADA model of the AEGIS radar scheduler.

Purdum, James H.

NAVAL POSTGRADUATE SCHOOL
Monterey, California



THESIS

AN ADA MODEL
OF THE
AEGIS RADAR SCHEDULER

by

James H. Purdum

December 1986

Thesis Advisor: Uno R. Kodres

Approved for public release; distribution is unlimited.

unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION: unclassified	1b RESTRICTIVE MARKINGS:
2a SECURITY CLASSIFICATION AUTHORITY:	3 DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE:	
4 PERFORMING ORGANIZATION REPORT NUMBER(S):	5 MONITORING ORGANIZATION REPORT NUMBER(S):
6a NAME OF PERFORMING ORGANIZATION: Naval Postgraduate School; 6b OFFICE SYMBOL (If applicable): 52	7a NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
6c ADDRESS (City, State, and ZIP Code): Monterey, California 93943-5000	7b ADDRESS (City, State, and ZIP Code): Monterey, California 93943-5000
8a NAME OF FUNDING/SPONSORING ORGANIZATION; 8b OFFICE SYMBOL (If applicable):	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER:
8c ADDRESS (City, State, and ZIP Code):	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO; TASK NO; WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)

AN ADA MODEL OF THE AEGIS RADAR SCHEDULER

12 PERSONAL AUTHOR(S) Purdum, James H.

13a TYPE OF REPORT: Master's Thesis	13b TIME COVERED FROM _____ TO _____	14 DATE OF REPORT (Year, Month, Day): 1986 December	15 PAGE COUNT: 181

16 SUPPLEMENTARY NOTATION

COSATI CODES		
FIELD	GROUP	SUB-GROUP

18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

ADA model of the AEGIS radar scheduler; NPS AEGIS project model

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This thesis presents a software implementation in JANUS/ADA of the Radar Scheduler process based on previous thesis work developed for the NPS AEGIS Control Program for a multi-microprocessor system. This thesis is a first effort in implementing the NPS AEGIS project model in JANUS/ADA. Included are the results of the preliminary real-time testing and logical tests of the Radar Scheduler module.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT: ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ OTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION: unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL: Prof. Uno R. Kodres	22b TELEPHONE (Include Area Code): (408) 646-2197; 22c OFFICE SYMBOL: Code 52Kr

FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE: unclassified

Approved for public release; distribution is unlimited.

An ADA Model of the AEGIS Radar Scheduler

by

James H. Purdum
Lieutenant, United States Navy
B.S., Arkansas State University, 1979

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1986

ABSTRACT

This thesis presents a software implementation in JANUS/ADA of the Radar Scheduler process based on previous thesis work developed for the NPS AEGIS Modeling Project. The project is an emulation of the AEGIS AN/SPY-1A Radar Control Program for a multi-microprocessor system. This thesis is a first effort in implementing the NPS AEGIS project model in JANUS/ADA. Included are the results of the preliminary real-time testing and logical tests of the Radar Scheduler module.

THESIS DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they can not be considered validated. Any application of these without additional verification is at the risk of the user.

Some terms used in this thesis are registered trademarks of commercial products. Rather than attempt to cite each occurrence of a trademark, all trademarks appearing in this thesis are listed below the firm holding the trademark:

1. U. S. Government (AJPO)
   a. ADA Programming Language
2. RR Software, Inc.
   a. JANUS/ADA Programming Language
3. Digital Research, Box 579, Pacific Grove, California
   a. PL/I-80 Programming Language
4. Intel Corporation, Santa Clara, California
   a. iSBC 86/12A Single Board Computer
5. Zenith Data Systems Corporation
   a. Z-100 Series Computer

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

The AEGIS Combat System (ACS) is an automated, rapid reaction shipboard combat system. As an automated combat system, it is designed to control shipboard sensors, manage electronic data processing, and assist in the making of time critical combat decisions while simultaneously engaging air, surface, and subsurface threats. Additionally an AEGIS platform posesses the capacity for defending its accompanying forces against the same threats. To meet this demanding environment the ACS relies on a specially designed computer system to assist in every phase of combat engagement.

At present the computer system is composed of three banks of four processors and up to three uni-processor AN/UYK-7 computer systems, totalling fifteen 32-bit processors. In addition, there are six or more AN/UYK-20 16-bit minicomputers in the system making the AEGIS system the largest network of computers dedicated to a combat system. The faculty and officer students at the Naval Postgraduate School have been interested in ways to reduce the costs of the ACS without compromising its capability. As a result the Naval Postgraduate School developed the AEGIS Modeling Group to study the problem. The group's major goal and approach to the problem is to model the AEGIS Combat System's computer suite using multi-microprocessor technology and the latest software engineering principles.

The feasibility of a multi-microprocessor approach was first addressed by Gayler's thesis [Ref. 1: p. 15]. Gayler explains that state of the art advances in microelectronics should be used as an alternate method of processor implementation in future AEGIS platforms. In order to realize the benefits of large scale integration (such as reduced size, weight, cost, and an increased reliability) a distributed multi-microprocessor architecture was proposed. Further studies into the hardware implementation were carried out in thesis work by Dilmore [Ref. 2: pp. 7-70] which describe hardware implementation for the NPS model. After the hardware decisions were made for the model, Riche and Williams [Ref. 3: pp. 11-232] laid out the design for the software foundation for the AN/SPY-1A radar control.

The NPS design of the software places a major emphasis on concurrent process management, both asynchronous and periodic. A Multi-Computer Real Time

Executive (MCORTEX) was developed in a sequence of six thesis projects to permit parallel processing in each computer in the multiprocessor system. At present, thesis work is being done to provide concurrent process management by creating an ADA language interface to the MCORTEX system. MCORTEX can then be used to coordinate the asynchronous processes of the ACS's model in a multi-microprocessor environment. The main objective of this thesis is to implement the Radar Scheduler process using the JANUS/ADA programming language. This in turn will provide a major portion of the overall model and the opportunity to study the feasibility of real-time programming in ADA.

Chapter II of this thesis presents the software documentation principles for the NPS model of the ACS. This includes a discussion of the Janus Ada programming language and an explanation of the program documentation scheme used by the NPS AEGIS Modeling Group. Chapter III describes the NPS model of the Radar Control System. A functional description is given for of each the Radar Control Group modules to be modeled. A more detailed description of the Radar Scheduler is provided in Chapter IV. Chapter IV presents the Radar Scheduler design and necessary documentation for the Radar Scheduler source code. This is the major section of this thesis which emphasizes the functional requirements, data structure specifications, and an explanation of the algorithms implemented by the Radar Scheduler program. Chapter V provides information on testing the algorithms implemented. This includes a discussion on the module testing philosophy for time critical programs and the strategy used in testing the Radar Scheduler modules. Chapter VI presents the conclusions and recommendations for further research.

II. SOFTWARE DOCUMENTATION

A. SOFTWARE ENGINEERING PHILOSOPHY

The primary goal of software engineering is to improve the quality of software products while increasing the productivity of software engineers. This goal applies as well to the AEGIS Modeling Group. Due to the continuing turn over and time constraints of research students, it is essential that this project be based on sound software engineering principles. In view of this the AEGIS Modeling Group has adopted the following philosophy. First, both the design and the source code must be clear, easy to understand, and modifiable. Second, the model must be constructed within the constraints of the AEGIS program specifications.

The first goal mentioned is achieved through a top down modular design, using structured programming. Moreover, the documentation must also emphasize this hierarchical structure with the upper-most level of documentation presenting the least detailed view and successive levels becoming more and more detailed. This way the reader is not overwhelmed by details and thus is more able to grasp the structure of the program.

The second goal relating to program specifications is very important. Unless the design is based on a firm knowledge of the program specification, the modeling effort is in vain. Each module's documentation will include a description of its purpose. All modules are constructed so that they obey the functional specifications of the AEGIS AN/SPY-1A Radar Controller software.

B. THE ADA PROGRAMMING LANGUAGE

The ADA programming language was designed in accordance with the requirements defined by the United States Department of Defense. In general, these requirements call for a language with considerable expressive power over a wide application domain [Ref. 4: p. 1-1]. Of particular interest in our application is the languages ability to cover real-time programming. To support real-time programming, ADA provides facilities to model parallel tasks and to handle exceptions. Unfortunately, the ADA language implementations do not provide runtime environments for multi-single-board computer based systems. In order to use the multiprocessor environment, the MCORTEX operating system is used to permit

parallel processing in the ADA language. In particular, use is made of the JANUS/ADA language, which differs from the fully validated ADA primarily because tasking and generics have not yet been implemented. Since the MCORTEX operating system is used for parallel processing, there is no need for the tasking features of the ADA language, and therefore JANUS/ADA is suitable as the programming language to implement the present version of the AEGIS model.

In the JANUS/ADA programming language modules are composed of packages that can be separately compiled. The package usually contains a specification and a body. Each of these parts reside in separate files for compilation purposes. Files containing the modules specification have the file type "LIB". Files containing the package body have been given the default JANUS/ADA file type "PKG".

ADA's enumeration types allow the user to add clarity to the code and program at a much higher level of abstraction. The clarity is achieved by creating data structures that can be easily read rather than having to decipher their purpose in the code.

C. PROGRAM DOCUMENTATION REQUIREMENTS

Documentation of the Radar Scheduler Model program in this thesis is in keeping with the PL/I-80 version which was modeled after the Software Requirements for the A7-E Aircraft document. The requirements call for documenting design decisions that will impact the future development of the program, what the functional requirements of the program are, what the interface data structures are composed of, how the program's internal data structures were fashioned, and the modular decomposition of the program. [Ref. 5: p. 23]

All the design decisions which were made for the Radar Scheduler Model can be found in Chapter 4 which serves as a module design document. Each Radar Scheduler module's design documentation contains the following:

1. A functional abstract description of the module,
2. Common memory interface,
    a. Data structures consumed,
    b. Data structures produced,
3. Internal process data structures,
    a. Data structures consumed,
    b. Data structures produced,

4. Local module data structures,
5. A description of the module in an algorithmic language.

Design decisions made for the purpose of testing the Radar Scheduler are documented in Chapter V.

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 7 Dec 86
-- MODULE TYPE: Skeleton Sample {Vers 1.0}
-- PURPOSE: Depict format for module source code
-- NAME: Sample ... fig2.pkg

-- This sample module source code skeleton represents
-- the format to be used in documenting modules.

WITH global.tables;   -- declare global data structures
                      -- and common memory
PACKAGE BODY fig2 IS

    USE global.tables;

    TYPE localvar IS INTEGER;  -- declare any local
    var: localvar;             -- module variables

    PROCEDURE sample(parameter: IN INTEGER) IS
        procvar: INTEGER;  -- declare procedure variables

    BEGIN
       -- code for the procedure
    END sample;

BEGIN
    -- If desired local or global variables
    -- can be initialized here.
    -- If this module was not designed for the
    -- procedure "sample" then the code for the
    -- main program "sample" would go here.
END fig2;
```

Figure 2.1 Source Code Documentation.

Code documentation must maintain a balance between verbosity and scarcity of comments. The prime objective is to increase readability and ease the task of maintenance. An example of the format for source code documentation in this thesis is given in Figure 2.1 . The source code documentation used in this thesis can be generally broken into three parts, a module header, a module description, and short comments for code explanations that may not be apparent to the reader.

The purpose of the module header is to introduce the module to the reader. The header's template format helps to insure that vital information is not overlooked, while

adding to the structured approach of the documentation. Each module header provides the reader with the following information:

1. Who the owner of the module is.
2. The date that the module was last altered.
3. The type of module and the current version number.
4. The purpose for which the module was written.
5. The English language equivalent name of the module and the file name it is stored under.

Following the module header is the module description. The module description is used to explain the general functional logic which controls the execution of the algorithm implemented by the module. Included in the module description are the input and output parameters used by the module.

The last type of documentation, short in code comments, are used to introduce the major functions within the module. The comments will describe what the function has been designed to do.

D. VARIABLE NAMING CONVENTIONS

A common pitfall of the applications programmer is "excessive contraction" of variable names. In keeping with the desire to maximize readability, the convention for naming variables is based on two principles. First, that the variable name be long enough to be unique to the language and the reader, and second, that the name reflect the variables purpose. However, if the lengths of the variable names are too long, it becomes difficult to organize the source code to fit on the standard 8.5 inch by 11 inch page. Consequently the programmer must consider the fact that he will not be the only one to read his code and decide accordingly.

In the case of this thesis variable naming was based on the PL/I-80 version names. The reason for this decision was to avoid confusion on the part of future researchers. This was necessary since the majority of the variables are defined by common memory interface data structures that were designed in the early stages of the Radar Control Group modeling process. Since these data structures act as interfaces to the other major processes, adding, changing, or deleting variables in these data structures was not a decision that should be made from the Radar Scheduler processes design level.

E. FILE NAMING CONVENTIONS

A file naming convention has been established in order to keep track of the large number of files that make up the the Radar Controller model. This convention has been preserved by this thesis in order to maintain consistance with the previous versions. The JANUS/ADA modules in this thesis have been named after their PL/I-80 predecessors so that their names will serve as a cross-referencing tool to facilitate future research.

TABLE 1

NPS AEGIS MODULE NAMING CONVENTION

MODULE NAME	CODE	FILE NAME
CONTROL GROUP		
Radar Scheduler	R	RRCM
Search Management	S	SRCM
Track Management	T	TRCM
Radar Return (input)	I	IRCM
Radar Output	O	ORCM
Channel I/O Control	H	HRCM
SUPPORT GROUP		
Switch Action and Display	A	ARCM
C&D User Services	C	CRCM
Beam Stabilization	B	BRCM
Detection Processing	D	DRCM
ECM/Clutter Processing	E	ERCM
Frequency Management	F	FRCM
Track Association	K	KRCM
Load Evaluation	L	LRCM
Missile Communications	M	MRCM
WCS User Services	W	WRCM
Cross-Gating	X	XRCM

The scheme for naming the files is presented in Table 1 . Each of the major processes is assigned a unique one letter process identifier code. This code followed by the letters "RCM", make up the file name which stands for "Radar Control Module". Each of these major processes is composed of several subordinate modules. The file names of the subordinate modules correspond to the initials of their parent process followed by a module number. For example, the first module within the Switch Action and Display process would have the file name "SADM1" which stands for Switch Action and Display Module 1.

If the modules are further subdivided into submodules and subroutines then they are uniquely identified through file name typing. That is, the filename takes on the parent process initials and module number followed by a submodule letter (A-Z). For instance, the Track File Initialization module, DPM1, which is part of the Detection Processing Module, has a subroutine which is used to place a node at the end of a linked list. Since this subroutine is part of DPM1, it has the file name DPM1A.

Some subroutines may be used by many modules. In this case they have more than one parent process and are classified as "Common Service Routines". These modules use the file name "CSR" followed by a number to uniquely identify them from other "Common Service Routines". For example, the subroutine "rand" , which generates a pseudo-random number, is shared by several modules, therefore, it has the file name "CSR5". There is also one other module which is shared by the major processes. This module contains global data structures and its file name is "GLOBAL".

The Radar Control Module also incorporates files that contain data structures which act as a "common memory interface" between the major processes. These common memory interfaces are called "tables" and their file names are the initials "TAB" followed by a number (0-77). The table's file names are organized into the matrix presented in Table 2 . This table shows data structures which interface with the major processes. The columns of the matrix contain entries which identify tables used as the input data structures to the process identified by the letter on top of the column. The rows of the matrix contain entries that identify the tables which are used as output data structures from the process identified by the letter in the left most column of the matrix. For example, "TAB3" is an interface data structure between the Radar Return process (I), and the Radar Scheduler process (R). To the Radar Return process "TAB3" is an input data structure and to the Radar Scheduler process it is an output data structure.

Since all the modules are defined as Ada packages, some modules may incorporate two files, the package specification and the package body. To distinguish between them, the file containing the specification has the file type "LIB". The file containing the body has the file type "PKG". For example, the files "RRCM.LIB" and "RRCM.PKG" are collectively the Radar Scheduler process.[1]

---

[1]Unless specified the words process, module, submodule, subroutine, etc., in this thesis refer to the entire package data structure.

TABLE 2

COMMON MEMORY INTERFACE MATRIX

	CONTROL GROUP							SUPPORT GROUP										
	R	S	T	P	I	O	H	A	C	B	D	E	F	K	L	M	W	X
R	0	1	2		3			4	5	6	7		8		9	10		
S	11	0						12			13	14			15		16	
T	17		0	18							7	19						
p	20				21			22	23	24	25	26		27		10	28	29
I						30	31											
O	32	33		34				35	36	37			38			10	39	
H	40					40												
A		41		42	43				44			45			46		47	
C		48		49	50			51						52	53			
B	54				55				56									
D					57			4										58
E					59													60
F	61							62									63	
K				64														
L	65	66		67							68							
M					10												69	
W				70	71						72							
X	73	74		75							76	77						

III. NPS MODEL OF THE AN/SPY-1A RADAR CONTROL GROUP

The NPS model of the AEGIS system is designed to emulate the Radar Control system. The overall system, as described by Gayler [Ref. 1: p. 1], contains seven subsystems:

1. Radar System AN/SPY-1A
2. Command and Decision System MARK 1 MOD 0
3. Weapons Control System MARK 1 MOD 0
4. Fire Control System MARK 99 MOD 1
5. Guided Missile Launching System MARK 26 MOD 5
6. Standard Missile
7. Operational Readiness Test System MARK 1

Research at NPS is focused on the first of these subsystems. The NPS AEGIS model implements a subset of the Radar Control Group subsystem, which in turn is a subset of the Radar System AN/SPY-1A. The reasons for this decision are discussed in Dilmore's thesis [Ref. 2: pp. 1-20].

The radar controller modules have been broken down into two major groups: control modules and support modules. The control modules are dedicated to the five major tasks of the radar controller:

1. Search Management,
2. Radar Scheduling,
3. Radar Output,
4. Radar Input and,
5. Track Processing.

The support modules perform more limited or specialized functions. Only the support modules required to implement the Radar Scheduler will be presented in this thesis.

A. RADAR CONTROL GROUP MODULES

This section gives an overall view of the Radar Control Group modules required for the modeling of the Radar Scheduler process. The first subsection describes the "control modules" and the second subsection is dedicated to the "support modules".

1. Control Modules

Four of the five "control" modules are required to model the Radar Scheduler functions: the Radar Scheduler process, the Search Management process, the Track Management process, and the Radar Return process. Of these control modules only the Radar Scheduler is fully modeled. The remaining control modules are at present only test harnesses for the Radar Scheduler process. The following description of these modules is based on previous thesis work [Refs. 3,5: pp. 43-50, 32-36].

a. Radar Scheduler

The Radar Scheduler's purpose is to schedule a mix of radar events. These events actually correspond to the transmission of a radar beam in a specific direction for a certain amount of time. In doing this the scheduler must ensure that a search volume of 360 degrees azimuth and 90 degrees elevation is covered. The term "mix of radar events" refers the various types of events such as search, track, missile control, etc.. Each of these events has a certain priority which must be taken into account to insure its timely occurance. The time limit for scheduling a mix of radar events in a given radar interval is 21 milliseconds. For this reason the Radar Scheduler is the most time critical function within the Radar Control Group to be modeled. A more detailed description of the module's functions will be given in the Radar Scheduler Design chapter.

b. Search Management

The Search Management module generates all the search type beam requests. The requests fall into three general categories, horizon search, above horizon search, and special request type beams. The beams are maintained in separate queues in accordance with the operational doctrine, the special request doctrine, and the radar event priority list.

The Search Management module must insure that enough horizon and above horizon requests are generated for a 360 degree azimuth and 90 degree search field. These two event types may be considered typical for normal operation.

On the other hand, special request beams correspond to events which would not be considered under normal searching operations. The special request category contains eleven radar event types, which enhance the radar search capability. The capabilities provided by the special request events are explained in the remainder of this subsection.

Two of the special request events are devoted to ECM Burnthrough. ECM Burnthrough is an electronic counter measure whereby the radar's transmission power is increased so that objects can be seen through jamming and/or clutter. Jamming is an electronic warfare technique used against radar to distort the radar return being viewed on the scope. Clutter is a term used to describe extraneous blips on the radar scope. The blips are usually caused by bad wether or sea conditions.

Special scans requests are also controlled by the Search Management process. Special scans include manual and moving target indicator (MTI) search requests. MTI is a technique that electronically filters out stationary objects from the radar display, thereby indicating which objects, or targets, are actually moving.

The Search Management module controlls passive search. In a passive search, unlike active search, the radar's transmitter is off and the radar's receiver is only used to detect the presence of another emmiter.

The extended search mode is another capability that is controlled by the Search Management process. This is used when there are several contacts along the same azimuth and the radar returns from the contacts become difficult to distinguish. Electronic blanking gates are inserted at the ranges of known contacts along the given azimuth so that the other contacts may be detected. The extended search mode may also be used to increase the search intensity in either range or bearing.

Horizon confirmation is an electronic means used to confirm targets in a clutter environment. Horizon confirmation and the clutter mapping process are both supported by the Search Management process.

The remaining special request functions supported by the Search Management module are missile and target acquisition requests, and partial dwell formatting of the requested beams. Modeling of the partial dwell formatting for use by the Radar Scheduler would require the use of classified documents. In order to maintain the unclassified status of this thesis, unclassified data has been substituted in the requested beam queues.

c. Track Management

The Track Management module is responsible for controlling all the requested track beams. Controlling the requested track beams entails three major tasks, with respect to both real and simulated targets.

The first task is track updating. Track updating entails an automatic smoothing of the position and rate of target return data. This data is used to predict

the future target positions with sufficient accuracy to maintain tracking. In order to accomplish this, the Track Management process must ensure that the highest priority tracks are given ample illumination time by the radar while simultaneously keeping lesser priority tracks illuminated within the constraints necessary to maintain their tracks. When the system becomes overloaded, the software must be able to compensate by eliminating those tracks that are perceived as least threatening. Naturally, this also implies that track processing must be able to evaluate new targets immediately so that special threat candidates and target splits can be identified.

The second task is track handling. Track handling includes the following:

1. Track dwell wave form selection,
2. Cross-gate preparation,
3. Transition from search to track mode,
4. Acceleration limiting processing, and
5. Automatic special mode processing.

The third task is special processing. Special processing is a collection of routines used for MTI tracks, missile tracks, and clock updates. Special processing also includes a routine for handling split targets. A split target is actually two or more targets that initially appear as one because they are so close together. When the target does split into multiple tracks each must be individually processed.

Special processing also includes a routine for processing Sensitivity Time Control (STC) data. STC is an electronic adjustment to the gain of the radar receiver based on the radar range to the target. This keeps objects closest to the radar, which provide very strong returns, from saturating the radar returns.

In conjunction with the three functions previously mentioned, the Track Management module is responsible for notifying Command and Decision (C & D) and Weapons Control Systems (WCS) of new tracks and any special threats.

d. Radar Return (Input)

The Radar Return module performs the initial processing on the raw data received from the SPY-1A Signal Processor. This includes data validity checks, clutter mapping, search processing, track angle error correction, and ECM analysis.

Once this is accomplished, the radar data must be routed to the appropriate modules for further processing. Evaluation and dissemination of this data is subject to the same 21 millisecond time constraint as the Radar Scheduler. In fact the Radar Scheduler relies on synchronization data from the Radar Return module to maintain scheduling interval synchronization with the radar hardware.

2. **Support Modules**

There are fifteen "support modules" contained in the Radar Control Group Modules. These support modules function as background processes to provide support for the "control modules". Seven of these modules are required for modeling the Radar Scheduler function. The description of these modules is based on information supplied in previous thesis research for the AEGIS Modeling Group [Refs. 5,3: pp. 36-40, 51-55].

a. Switch Action and Display

The Switch Action and Display module acts as an interface between the Radar System Controller (RSC) and the radar. This allows the RCS to monitor and control the radar in accordance with operational doctrine and the individual desires of the operator.

The module provides information to the Radar Scheduler function which is used in formatting selected beams for the current scheduling interval. The information consists of inhibited radiation regions and the radiation power level (low or high). Radiation inhibit regions are regions where doctrine control does not allow radar transmission. The regions are defined via start and stop bearings.

b. Command and Decision User Services

The Command and Decision User Services module acts as an interface between the Radar Control Group and the Command and Decision Group. The purpose of the module is to check and route all the messages between these programs. To accomplish this, a priority level message scheme is used in order to meet the required track report interval rates and other report rates during peak radar load periods.

c. Beam Stabilization

The Beam Stabilization module performs two main functions. First it transforms the ship's motion matrix (roll, pitch, and yaw) into a stable space matrix which is used for radar beam guidance. Second, it assigns the array face limits which are used in formatting scheduled dwells.

The fore and aft gyro data converters supply the necessary roll, pitch, and yaw information along with ship's heading for stabilization. This information is used to compute the stable deck space matrix.

The ship's motion matrix and the bearing limits for the four phased array antennas are passed to the Radar Scheduler. The Radar Scheduler then uses the information to format the selected dwells.

d. Detection Processing

The Detection Processing module is executed only when target or beacon detection data is received from the Radar Return module. When this occurs the module looks for detections as a result of normal search (clear and MTI), extended range search, horizon confirmation, missile acquisition, and target acquisition dwells. The module is also responsible for initiating track requests and preparing detections for cross-gating.

The Track File data is maintained by the Detection Processing module. The Track file is a linked list data structure capable of handling as many targets as can be tracked within the hardware constraints of the computer. Access to the Track File is provided to the Radar Scheduler by the Track Management module via an index into the file.

e. Frequency Management

The Frequency Management module is used to assign radar frequencies to the dwells selected by the Radar Scheduler during each scheduling interval. The frequencies and phase codes assigned to the dwell are made to conform with sector and global doctrines.

In accordance with the doctrines, frequency channel selections are in one of three modes, fixed, random, or prelook. Fixed implies one designated channel only. The random mode means that frequencies are selected on a random basis from the available channels established by the current doctrine. Prelook implies frequency selection of the least jammed channel frequency based on the results of a passive angle track (PAT) frequency analysis. By use of these frequency channel selection modes the Frequency Management module can generate the Radar Scheduler input data for the frequency operating channel and subpulse-frequency order selection.

The phase code to be used within each dwell is also specified by this module. The phase codes are selected at random from a list of the phase codes available during the current schedule for each dwell. A separate list of phase codes are maintained for clear and MTI type dwells.

Each scheduled dwell will contain channel frequency, band frequency, phase codes, and inhibited frequency channels. If the dwell is an MTI type then it will also contain MTI frequency data. If the dwell is a missile type then missile uplink and downlink frequencies will be included.

f. Load Evaluation

The Load Evaluation module provides system's load information to the Command and Decision element as well as the Radar System Controller. The systems load is based on seven indicies which must be computed periodically. These seven indicies are:

1. Transmitter utilization load,
2. Control group track file load,
3. Control group computer processing load,
4. Search frame time for horizon and above horizon scans,
5. Track time,
6. Track transition index, and
7. Radar time utilization index.

The track time variable is used by the Radar Scheduler as input to reset its' track time counter. The Radar Scheduler keeps a running total of track time. The Track time index provides the percentage of radar usage dedicated to tracking targets over an average five second time span.

g. Missile Communication

The Missile Communication module provides an interface between the Weapons Control System guidance command generation and Radar Control Program's guidance control link. Uplink missile guidance commands and the missile identification code are sent to the Radar Scheduler. Further information on the operation of this function can be found in classified documents.

IV. RADAR SCHEDULER DESIGN

The design of the Radar Scheduler function is based on the PL/I-80 version implemented by Grant, [Ref. 5: p. 41]. This chapter presents the design details for the JANUS/ADA version of the Radar Scheduler process.

A. PURPOSE OF RADAR SCHEDULER

The Radar Scheduler's purpose is to schedule a mix of radar events (dwells) in a way that ensures the occurrence of necessary events in a timely manner. The various types of radar events used in this model were taken from open literature, rather than the actual classified list found in the AEGIS performance specifications. [Ref. 5: p 41] The list of radar events shown in Table 3 are the radar events used in this model.

TABLE 3

RADAR EVENT PRIORITY LIST

PRIORITY	RADAR EVENT	QUEUE IDENTITY
1	ECM Burnthrough	Special Request
2	Target Definition	Special Request
3	Special Test	Special Request
4	Engaged Hostile Target	Track
5	Own SM-2 Missile Guidance	Missile
6	Pre-Engaged Hostile Target	Track
7	High Priority Track Transition	Track
8	High Priority Track Confirmation	Track
9	Horizon Search	Search
10	Special ECM Burnthrough	Special Request
11	Special Target Definition	Special Request
12	Special Scans (MTI,Man,etc)	Special Request
13	Special Target Acquisition	Special Request
14	Confirmed Hostile Track	Track
15	Assumed Hostile Track	Track
16	Unevaluated Track	Track
17	Controlled Friendly Track	Track
18	Track Confirmation	Track
19	Track Transition	Track
20	Assumed Friendly Track	Track
21	Confirmed Friendly Track	Track
22	Above Horizon Search	Search
23	Above Horizon Search Test	Special Request
24	Simulation Dwell	Special Request
25	Diagnostic Dwell	Special Request
26	Dummy Dwell	Special Request

The actual mix of radar events that get scheduled is determined by the Radar Scheduler. The schedule is governed by the radar resources available, time constraints, and an event's priority relative to the other events that are being requested. The details of the process are explained in the following section. The Radar Scheduler's algorithmic description is given Figure 4.1 .

```
Begin Radar Scheduler;
    For number of intervals loop;
        Advance radar loop event counts;
        Get value of real time (CSR7);
        Swap external dwell buffers (RSM2);
        Do enhancement of Radar Event Priorities (RSM3);
        Resynchronize computer and radar times (RSM4);
        Begin beam selection
            Traverse the Priority List
                Traverse the event queue
                If search or special request queue then
                    Put beam information in Control Table;
                    Do Supplementary Dwell Processing (RSM6);
                    Do Face Assignment (RSM7);
                    Do Radar Doctrine (RSM8);
                    Satisfy Hardware Constraints (RSM9);
                    If satisfied then
                        Fill External Tables (RSM10);
                        Insert Dwell in Control Table list (RSM5);
                    Else
                        Delete Dwell from Control Table list (RSM5);
                Else
                    Put beam information in Control Table;
                    Do Supplementary Dwell Processing (RSM6);
                    Do Face Assignment (RSM7);
                    Do Radar Doctrine (RSM8);
                    Satisfy Hardware Constraints (RSM9);
                    If satisfied then
                        Fill External Tables (RSM10);
                        Insert Dwell in Control Table list (RSM5);
                    Else
                        Delete Dwell from Control Table list (RSM5);
                End Traverse the Event Queue;
                Compute Elapsed Time (RSM12);
            End Traverse Priority List;
        End Beam Selection;
        If interval results are to be displayed then
            Display Interval Scheduling Results (RSM13);
        Free Control Table Memory for Next Interval (RSM14);
    End For number of intervals loop;
End Radar Scheduler;
```

Figure 4.1 Radar Scheduler Algorithm.

B. FUNCTIONAL DESCRIPTION OF RADAR SCHEDULER

The Radar Scheduler operates in a continuous loop selecting requested beams, generated by the Search and Track management processes, for scheduling. Once selected, the beams are formated into dwells. If the hardware and other constraints are satisfied, then scheduled dwells are sent to the Radar Output function and are packed into the Channel Output Buffer for transmission to the Signal Processor. The time it takes to complete this task is defined as a scheduling interval. Furthermore, the median scheduling interval is defined to be 21 bms (binary milliseconds).

The requested beams are stored for use by the Radar Scheduler in four types of queues. The search and special request type queues are maintained by the Search Management Process.[2] The track and missile type queues are maintained by the Track Management Process.[3] The Radar Scheduler gains access to these queues through pointers in the Priority Event List. The Priority Event List in turn provides the scheduler with a mechanism for prioritizing the requested beams (see Table 3). Therefore, an ordering of the beams is developed in two ways. From the queue structure they are ordered in a first come first served fashion. Second, the queues themselves are ordered by association with the Priority Event List. The queue associated with the radar event at the beginning of the list has the highest priority as indicated in the table. Together these data structures provide the basis of the priority scheme used for selecting the requested beams.

When the Radar Scheduler program is loaded for execution, the Priority Event List and the scheduler's internal data structures are automatically initialized. Also included in this initialization are the data structures that the scheduler produces.

The scheduling interval begins after advancing the event counts and updating the real time. Next the external dwell buffers are swapped preparing them for the next set of dwells to be scheduled. Once the buffers have been swapped, the enhancement procedure modifies the Priority Event List holding the requested beams.

The enhancement routine dynamically enhances the priority of the events in the list and in effect changes its traversal order. Changing the traversal order of the events increases the probability of selection for those requested beam queues associated with the events at the head of the list.

---

[2]Search and special request queues are formed from the Search Node data structure.

[3]Track and missile queues are formed from the Track Node data structure.

After enhancement and resynchronization of computer and radar times, the beam selection process begins. During this process beams are considered for selection by traversing the Priority Event List in its' current priority order. The highest priority beam is selected first and so on until the resource constraints are exhausted for that particular interval.

There are several resource constraints that must be considered. There must be sufficient radar resources available to handle the requested beams. The elapsed time for the scheduling interval must be less than the allowed elapsed time. The total dwells scheduled must not exceed the maximum allowed during an interval. The final constraint occurs by completely traversing the Priority Event List and in effect terminating the beam selection process. If these constraints are met, traversal of the Priority Event List is continued.

If the event's beam queue is not empty then the beam queue is traversed processing each requested beam. First the beams information is inserted into a Radar Event Control Table node. Next supplementary dwell processing occurs in preparation for scheduling. After this is accomplished, the selected beams are given an array face assignment and a comparison is done between the beam's transmission and the radar doctrines that are in effect. Once this is accomplished the beams must meet the final selection criterion of satisfying the hardware constraints.

If the hardware constraints were satisfied, then the selected beam is sent to the external dwell buffers, load evaluation occurs, and the node is added to the Radar Event Control Table. If the hardware constraints were not satisfied, then the node is returned to the pool of available Radar Event Control Table nodes for future use. The Radar Scheduler then considers the next beam in the event's request queue, and so on until the queue is empty.

When the beams in the queue have been considered the scheduler moves on to the next event in the list. This continues until all the events in the Priority Event List have been considered or the resource constraints can no longer be met. Once this occurs, the scheduling interval is completed and the elapsed time is computed.

If the results of this interval are requested by the program operator, then the appropriate information is dumped into a file called "RSOUT.TXT". Then the memory is freed up for the next scheduling interval by returning the nodes that make up the Radar Event Control Table to the pool of available nodes.

The run-time Radar Scheduler model does not implement the following design time modules: Resynchronization (RSM4), Supplementary Dwell Processing (RSM6), Radar Array Face Assignment (RSM7), Comply With Radar Doctrine (RSM8), and the Radar Load Evaluation (RSM11) module. Implementation of the modules that are not coded would require knowledge of the required data structure values to be produced. Due to time constraints, the design decisions which must be made to produce those values must be deferred until the processes that will use them are designed and implemented. [Ref. 5: pp. 46-47]

C. EXTERNAL DATA STRUCTURES

The external data structures are global constants, variables, and type declarations which act as interfaces between the Radar Scheduler modules and the other Control Group and Support Group modules. The modules are referred to as tables and are numbered consecutively from zero to seventy-seven.

A Common Memory Interface Matrix, Table 2, is designed to be used as an index into the listing of the external data structures located in Appendix D. The applicable data structures for the Radar Scheduler Module can be found by reading across row "R" to locate the output data structures and down column "R" to locate the input data structures.

1. Data Structures Consumed

These data structures reside in common memory and are declared by Library Packages for use by the Radar Control processes which read from and write to them. The Radar Scheduler uses the data structures in the following tables to select and format dwells during a scheduling interval.

a. Table 0 - Priority Event List

The Radar Event Priority List is visible to the Radar Scheduler (consumer), Search Management (producer), and the Track Management (producer) procedures only. The variable **pri_lst** is a one dimensional array of radar events which correspond to the Radar Event Priority List depicted by Table 3 . The array simulates a linked list so that priorities can be changed dynamically during execution. Each element in the array is a pointer corresponding to a unique Radar Event. Each event contains a **BeamQue** which is a variable record that holds a pointer to a queue of requested beams. Although each queue is unique, they are classified as either search, special request, track, or missile type queues, depending which Radar Event the beam queue

belongs to. The variable record is designed to associate search and special request beams with the search node data structure found in TAB1.LIB. The track and missile type beams are associated with the track node data structure of TAB2.LIB. The Priority Event List is the key to the selection process. By placing the beam queues in the Priority Event List, the requested beams are prioritized by association with their respective radar events. The requested beam's priority is used as the main criterion for selection. Furthermore, since the Priority Event List is designed to act like a linked list, the traversal order can be changed for each interval to ensure the eventual selection of the lowest priority beams.

The constants and variables defined by this data structure are described as follows:

- **status** is a boolean flag which is set to true when the event queue has information in it.
- **eventnm** is a string with the event name corresponding to the Radar Event Priority List (Table 2).
- **max_nodes** is a variable representing the maximum number of nodes that the event's requested beam queue can hold.
- **que_id** is an enumeration type variable corresponding to one of four possible queue types, **search**, **special_request**, **track**, or **missile**.
- **que_ptr** is actually a variable record containing a pointer to the first node in the event's requested beam queue. If the **que_id** is a search or special request type queue then the pointer **Snode** is visible, otherwise the pointer **Tnode** is visible.
- **enhnc** acts as a flag which when set to true indicates that the priority of the event can be enhanced by the Radar Scheduler when the conditions stated in Radar Scheduler Module 3 are met.
- **b_pri** is an integer variable which holds a constant value corresponding to the event's base priority. This value is the same as the "pri_lst" array's index value.
- **c_pri** is an integer variable which corresponds to the event's current priority as determined by Radar Scheduler Module 3.
- **ltx** indicates the last time a beam from this event was selected by the Radar Scheduler.
- **allwd_ltx** indicates the allowed time between selection for this type of event.
- **slct_flg** is a flag which is set to true if a beam from this event was selected during the previous interval.
- **nxt** is an integer variable used as an index or pointer to the next priority in the **pri_lst**. It's value can only be changed during priority enhancement. The value "0" acts as an end of list marker in the same way that a null pointer marks the end of a linked list.
- **low_enhnc** is a constant which stands for the lowest enhancement value, 4.
- **max_pri** is a constant which stands for maximum priority but is actually the maximum number of events in the Priority Event List or the length of the **pri_lst** array, 26.

b. Table 1 - Search Node

This data structure acts as a template for the nodes within the search and special request beam queues. Table 1 also provides an interface for the Radar Scheduler process and the Search Management process. The fields contained in the search node record are as follows:

- **info.mode** is the event number identifier. It can have a unique value between 1 and 26.
- **info.bid** is a character string which acts as a unique beam identifier for scheduler efficiency analysis.
- **info.beam_posit.azim** is an integer variable describing the requested beam's azimuth position in deck space coordinates.
- **info.beam_posit.elev** is an integer variable describing the requested beam's elevation position in deck space coordinates.
- **info.inst_rge** is a variable which gives an indication of the range to which the search beam is to be effective.
- **info.stc_data** is the sensitivity time control variable. This is used as an indication of how much power will be allowed to be used to pull contacts out of clutter.
- **info.doct_blnk_gte** holds the value for the radar doctrine range gate.
- **info.cltr_blnk_gte** holds the value for the radar clutter range gate.
- **info.eof_indic** is a flag which when set to true indicates that search frame has been completed. A search frame consists of 360 degrees of azimuth and 90 degrees of elevation.
- **info.req_id** is used to hold the identity of the requesting process for special request beams.
- **nxt** is a pointer to the next node in this queue, which is a linked list of Search Nodes.

c. Table 2 - Track Node

This is a common memory interface data structure between the Radar Scheduler process (consumer) and the Radar Return process (producer) only. The track Node is a template for the nodes within track and missile beam queues. The data fields declared in the Track Node are:

- **info.mode** is the event number identifier. It can have a unique value between 1 and 26.
- **info.bid** is a character string which acts as a unique beam identifier for scheduler efficiency analysis.
- **info.p_trk** is a pointer to the Track File which this node is requesting a beam for. The Track File data structure is located in Table 7.
- **nxt** is a pointer to the next Track Node in this queue, which is a linked list of Track Nodes.

d. Table 3 - I to R

Table 3 is a common memory interface data structure for the Radar Scheduler (consumer) and Radar Return (producer) processes. The data structure the following SPY-1 radar status variables:

- **resynch_time** is the amount of time that the Radar Scheduler is out of synchronization with the SPY-1 radar expressed in milliseconds.
- **loop_con** is an integer variable used to tell the Radar Scheduler how far ahead or behind in the radar control loop the Radar Control Program is.
- **xmsn_status** is a boolean variable indicating the transmission status of the previous interval's scheduled beams.

e. Table 4 - A to R

Table 4 is a common interface data structure between the Detection Processing Process (consumer), Radar Scheduler Process (consumer), and the Switch Action and Display Process (producer). The data structure contains RCS commands to aid in the formating of the radar dwell buffer. The data fields are as follows:

- **power_flg** is a boolean variable that represents the status of the radar power. When "power_flg" is set to true, the power is to high, and when it is set to false, the power is to low.
- **rad_pwr_option** is a boolean variable that represents the status of the radar power option. When set to true the power option is on, and when set to false the power option is off.
- **rad_inhibt_regions** is a five element array of records containing start and stop bearings for one of five possible radiation inhibition regions definable by doctrine.
- **rad_inhibt_regions( ).start_bng** is the bearing start indicator.
- **rad_inhibt_regions( ).stop_bng** is the bearing stop indicator.
- **op_doct.mtrks** is the maximum number of tracks to be initialized in the Track File. This value is used to aid in testing the Radar Scheduler Module.
- **op_doct.mintvls** is the maximum number of scheduleing intervals to be run on a test of the Radar Scheduler Model.
- **op_doct.dply_rect** is an integer variable used to define the interval display delay period.

f. Table 5 - Command and Decision User Services Interface

Table 5 is visible to the Radar Scheduler Process (consumer) and the Command and Decision User Services Process (producer). The interface contains one variable to communicate the transmission status:

- **rdr_silence** is a boolean variable which when set to true, informs the Radar Scheduler that radar silence is in effect.

g. Table 6 - Beam Stabilization Interface

Table 6 is a common memory interface between the Radar Scheduler Process (consumer) and the Beam Stabilization Process (producer). The data structure contains information concerning array face limits and the ship's motion matrix. The data fields are as follows:

- **face_antenna_limits** is a four element array of records containing the left and right for each of the four phased array antenna faces.
- **face_antenna_limits( ).limit_left** indicates the left most bearing limit to an accuracy of 0.1 degrees for the phased array antenna face indicated by the array index. The limits are converted into stable space bearings from deck space bearings by the gyro converters.
- **face_antenna_limits( ).right_limit** is the same as above only pertaining to the right most limit.
- **ships_motion_matrix.x_ship_dot** is the value of the ship's velocity in the x deck direction.
- **ships_motion_matrix.y_ship_dot** is the value of the ship's velocity in the y deck direction.
- **ships_motion_matrix.roll** is the amount of roll the ship is experiencing.
- **ships_motion_matrix.pitch** is the amount of pitch the ship is experiencing.
- **ships_motion_matrix.yaw** is the amount of yaw the ship is experiencing.
- **azim_limit_slope** is the limiting value of the rise in elevation for a given azimuth.

h. Table 7 - Track File

Table 7 is a common memory interface between the Radar Scheduler Process (consumer/producer), the Detection Processing Process (producer), Track Processing Process (producer/consumer), and the Track Management Process (consumer). The Track File acts as a memory map which allows dynamic allocation of memory for tracks as they are acquired. The files are arranged in a linked list of track file nodes. The track file nodes contain two major hierarchy levels beam data and track data. These data fields are defined as follows:

- **beam_data.mode** is an integer between 1 and 26 which corresponds to the identity of the radar event for this track.
- **beam_data.bid** is a character string which uniquely identifies the requested beam for this track.
- **beam_data.priority** is an integer value corresponding to the relative priority of this track compared to the other tracks in this Track File.
- **beam_data.posit.x** is the rectangular X coordinate of the track position as found when it was last illuminated by the radar.
- **beam_data.posit.y** is the rectangular Y coordinate of the track position as found when it was last illuminated by the radar.
- **beam_data.posit.z** is the rectangular Z coordinate (elevation) of the track position as found when it was last illuminated by the radar.

- **beam_data.posit.slnt_rge** is the straight line range to the track's last position.
- **beam_data.posit.x_dot** is the track's velocity in the X direction.
- **beam_data.posit.y_dot** is the track's velocity in the Y direction.
- **beam_data.posit.z_dot** is the track's velocity in the z direction.
- **beam_data.posit.rge_dot** is the tracks relative change in straight line range.
- **beam_data.trk_xsitn_flag** is a boolean variable which when set to true indicates that the track has a track historyless than or equal to four detections.
- **beam_data.ctl_grp_trk_num** is a number which corresponds to the track's Radar Control Group track number.
- **beam_data.ctsl** is a number used to historically record the track.
- **beam_data.xgte_bin_num** is a number which corresponds to the track's location in the cross-gating matrix.
- **beam_data.pred_azim** is a bearing value corresponding to where the track is predicted to be by the Radar Scheduler upon scheduling a track dwell on this track.
- **beam_data.pred_elev** is the predicted elevation of the track.
- **beam_data.low_elev_trk_flg** is set to true when the track's elevation is below a preset altitude. The actual settings are classified.
- **beam_data.log_ampld_est** is an estimate of the return signal strength of the radar echo bounced off this track.
- **beam_data.xmit_req_flg** is a boolean variable which when set to true indicates Track Processing is requesting a dwell be transmitted for this track.
- **beam_data.sim_tgt_flg** is a boolean variable which is set to true when the track is a simulated target.
- **nxt_trk** is a pointer to the next track node in the Track File. The data fields associated with this pointer are defined by the declarations in Table 2.

i. Table 8 - Frequency Management Interface

Table 8 is used by the Radar Scheduler Process (consumer) to complete formatting of selected radar dwells. The data structure contains frequency and waveform information in the following fields:

- **waveform** is an array from 1 to "max_dwells" of records containing waveform information.
- **waveform( ).freq_chnl** is the channel frequency to be used by this beam.
- **waveform( ).freq_band** is the frequency band to be used by this beam.
- **waveform( ).phase1_code** is the frequency phase code used for transmitting this beam.
- **waveform( ).phase2_code** is the second half of the frequency phase code used for transmitting this beam.
- **waveform( ).mti.pri** is the PRI code used when the beam is an MTI beam.
- **waveform( ).mti.dwl_length** is the length or duration of the dwell in microseconds if the beam is a MTI beam.

- **waveform( ).msle.uplnk_freq** is the frequency used to communicate with own ship's SM-2 missile, if the beam is a missile communication beam.
- **waveform( ).msle.dwn_freq** is the frequency used by the missile to transmit information back to the ship, when the beam is a missile communication beam.
- **waveform( ).inhib_freq_chnl** are the frequency channels which are prohibited to this beam.

j. Table 9 - L to R

Table 9 is the common memory interface between the Radar Scheduler Process (consumer) and the Load Evaluation Process (producer). This data structure is used by the Radar Scheduler Process to format track dwells and contains only one data field:

- **trk_tim_ctr_reset** is a boolean variable which when set to true informs the Radar Scheduler Process to zero out its track time counter.

k. Table 10 - Missile Downlink Interface

Table 10 is the common memory data structure between the Radar Scheduler Process (consumer) and the Missile Communications Process (producer). Since the actual data fields are classified, the downlink message is arbitrarily separated into eight parts as follows:

- **mssl_dwnlnk** is an array form one to "num_mssl_msgs" of records containing missile messages.
- **mssl_dwnlnk( ).prt_one** is an integer variable.
- **mssl_dwnlnk( ).prt_two** is an integer variable.
- **mssl_dwnlnk( ).prt_three** is an integer variable.
- **mssl_dwnlnk( ).prt_four** is an integer variable.
- **mssl_dwnlnk( ).prt_five** is an integer variable.
- **mssl_dwnlnk( ).prt_six** is an integer variable.
- **mssl_dwnlnk( ).prt_seven** is an integer variable.
- **mssl_dwnlnk( ).prt_eight** is an integer variable.

2. Data Structures Produced

These data structures are located in common memory. They are used by the Radar Scheduler Process (producer) and those processes which are consumers of the data.

a. Table 11 - Search Management Interface

Table 11 is the common memory data structure interface between the Radar Scheduler Process (producer) and the Search Management Process (consumer). The data structure is used by the Search Management Process to distinguish which

search queues require filling. The Radar Scheduler sets the flags when it has used a set number of search request beams. The flags are defined below:

- **hs_que_replen** is a boolean variable which when set to true indicates that the horizon search queue needs to be replenished.
- **ahs_que_replen** is a boolean variable which when set to true indicates that the above horizon search queue needs to be replenished.

b. Table 17 - Track Management Interface

Table 17 is the common memory data structure between the Radar Scheduler Process (producer) and the Track Management Process (consumer). The Radar Scheduler Process uses the data structure to hold the scheduled track dwells selected for transmission. The Track Management Process uses the data structure to asses its efficiency in prioritizing previously requested tracks. The data fields are defined as:

- sched_trk_beam_list is an array from one to "max_dwells" of records.
- sched_trk_beam_list( ).mode_id is a integer variable used to identify the radar event associated with the scheduled track.
- sched_trk_beam_list( ).trk_file_num is the unique track number of the scheduled track.
- sched_trk_beam_list( ).priority.

c. Table 20 - Track Processing Interface

Table 20 is a common memory interface used by the Radar Scheduler Process (producer) and the Track Management Process (consumer). The data structure contains information on each of the scheduled dwells for one scheduling interval. As dwells are formatted for transmission, the Radar Scheduler Process transmits the information. The data fields are defined as follows:

- **stble_spc_pos** is an array from one to "max_dwells" of records containing stable space position information.
- **stble_spc_pos( ).azim** is the bearing of the scheduled dwell in tenths of degrees.
- **stble_spc_pos( ).elev** is the elevation of the scheduled dwell in tenths of degrees.
- **stble_spc_pos( ).x_posit** is the rectangular X coordinate of the scheduled dwell.
- **stble_spc_pos( ).y_posit** is the rectangular Y coordinate of the scheduled dwell.
- **stble_spc_pos( ).z_posit** is the rectangular Z coordinate of the scheduled dwell.
- **sched_xmsn_time** is the scheduled transmission time in milliseconds for the scheduled dwell.

d. Table 32 - Radar Output Interface

Table 32 is the common memory interface between the Radar Scheduler Process (producer) and the Radar Output Process (consumer). It acts as a buffer for

the formatted dwells, and supplies information to the Radar Output Process to aid in completing the channel I/O control buffer. The data structures in Table 32 are defined as follows:

- **ptr_r_to_o** is a two element array of pointers used to access two linked lists of records containing dwell data.
- **ptr_r_to_o( ).dwl_data.mode** is the major transmission mode used by this dwell.
- **ptr_r_to_o( ).dwl_data.face** is the antenna face assigned to this dwell.
- **ptr_r_to_o( ).dwl_data.sub_mode** is a two element array of integers which are the frequency sub-modes assigned to this dwell.
- **ptr_r_to_o( ).dwl_data.dwl_idx** is the dwell index number for the current scheduling interval.
- **ptr_r_to_o( ).dwl_data.beam_purpose** is an integer which corresponds to the beam's mode value which indicates what purpose the dwell was scheduled for.
- **ptr_r_to_o( ).dwl_data.dwl_start_idx** is a 32 bit number which is the transmission time for the dwell to an accuracy which is classified.
- **ptr_r_to_o( ).dwl_data.doct_unblnk_gates** data has been read from the Search Management provided data.
- **ptr_r_to_o( ).dwl_data.clutter_unblnk_gates** data is the same form as doct_unblnk_gates.
- ptr_r_to_o( ).link is a pointer to the next dwell data record in the linked list.

e. Table 40 - Output Control Channel Buffer

Table 40 is a common memory interface between the Radar Scheduler Process (producer), the Radar Output Process (producer), the Radar Control Program, and the Radar Channel Control Program (consumer). The Radar Scheduler Process has responsibility for filling only a small portion of the data fields which are defined below:

- **occb_ptr** is a two element array of pointers to two linked lists of records containing Output Channel Control Buffer information.
- **occb_ptr( ).oa.cntrl_word.rdr_xmsn_on** is a boolean variable which when set to true indicates that the dwell is to be transmitted.
- **occb_ptr( ).om.face** is the antenna face assigned to this dwell.
- **occb_ptr( ).oh.pri1_mti** is the MTI PRI code used by the dwell when it is an MTI dwell.
- **occb_ptr( ).oh.pri2_mti** is the second half of the PRI code.
- **occb_ptr( ).ot** is a twelve element array of records emulating missile communications links.
- **occb_ptr( ).ot( ).otmsb** emulates a missile communication link when the dwell is a missile dwell.
- **occb_ptr( ).ot( ).otlsb** also emulates a missile communications link.
- **occb_ptr( ).ol.xmit_freq** is the transmission frequency used by this dwell.
- **occb_ptr( ).ol.rcm_freq** is the receiver frequency used to receive the return dwell signal.

- **occb_ptr( ).oj.subchnl_freq_group** is the group of sub-channel frequencies used to transmit this dwell.
- **occb_ptr( ).o_f.phse1_code** is the first part of the dwell frequency phase codes.
- **occb_ptr( ).og.fdbk1** is the first part of the dwell feedback phase codes.
- **occb_ptr( ).oa.cntrl_word.freq_group_slct** is a boolean flag which when set to true indicates the selected frequency group for this dwell.
- **occb_ptr( ).oi.dwl_1_start_time** is the start time for the dwell in microseconds.
- **occb_ptr( ).ob.detect1_thrsld** holds the value of an expected detection range for a track type dwell.
- occb_ptr( ).ob.detect2_thrsld holds the value of an expected detection range for a track type dwell.
- occb_ptr( ).ob.detect3_thrsld holds the value of an expected detection range for a track type dwell.
- **occb_ptr( ).oe.trunc1_thrsld** is a truncation signal level threshold value to assist in detecting targets.
- occb_ptr( ).oe.trunc2_thrsld is a truncation signal level threshold value to assist in detecting targets.
- occb_ptr( ).oq.elev_sector is the sector elevation limits for this dwell.
- occb_ptr( ).os.dply_azim is the bearing the dwell is to be transmitted on.
- occb_ptr( ).o_r.dply_elev is the elevation this dwell is to be transmitted on in degrees.
- occb_ptr( ).os.video_extnt is the signal level expected from this dwell.
- **occb_ptr( ).trk_gate_strt** is the starting range for gating a track dwell.
- **occb_ptr( ).link** is a pointer to the next Output Channel Control Buffer node available.

D. INTERNAL DATA STRUCTURES

All data structures that are internal to the Radar Scheduler Process and that must be shared by subordinate Radar Scheduler modules are located in the package specification of RSM0. The data structures are defined as follows:

- **rdrint** is a constant that represents the maximum time which can be spent in the Beam Selection Module.
- **srch_que** is a constant used to represent the event type Search.
- **sr_que** is a constant used to represent the event type Special Request.
- **trk_que** is a constant used to represent the event type Track.
- **mssl_que** is a constant used to represent the event type Missile.
- **srch_dwls** is a constant equal to the maximum number of Search events that can be scheduled in one interval.
- **sr_dwls** is a constant equal to the maximum number of Special Request events that can be scheduled in one interval.

- **trk_dwls** is a constant equal to the maximum number of Track events that can be scheduled in one interval.
- **mssl_dwls** is a constant equal to the maximum number of Missile events that can be scheduled in one interval.
- **rdr_rsrcs** is a constant which represents the percentage of radar resources which can be used in scheduling beams for an interval.
- **ppl** is an integer variable used as an index for traversing the priority list.
- **intvl_num** is an integer variable used to represent the number of intervals that have been executed.

The next data structure contained in RSM0 is the Radar Event Control Table. This is the primary data structure used by the Radar Scheduler to schedule a mix of radar events for one scheduling interval. The Radar Event Control Table is a linked list of records. The records which act as nodes contain the following fields:

- **srch_dwl** is a pointer to a Search Node data structure described in external Table 1.
- **trk_dwl** is a pointer to a Track Node data structure described in external Table 7.
- **ocb_data** is a record containing the data to be transmitted to the external dwell buffers. Explanations for the data fields in this record can be found in Sections IV.C.2.d and e.
- **beamid** is a 1 character string used for testing the logic of the Radar Scheduler algorithm.
- **dru** is an integer also used for testing the logic of the Radar Scheduler algorithm.
- **nxt_event** is a pointer which acts as a link to the next Radar Control Table node in the linked list.
- **rect** is a pointer to the head of the Radar Control Table's linked list data structure.
- **sp** is a pointer to the Search Node data structure described in Table 1.
- **tp** is a pointer to the Track Node data structure described in Table 2.
- **rect_ptr** is a pointer to the head of the linked list which makes up the Radar Event Control Table.
- **pool_ptr** is a pointer to Radar Event Control Table record.
- **buff_ptr** is a pointer to a the data structure described in Table 40.
- **cm_ptr** is a pointer to the data structure described in Table 32.

E. RADAR SCHEDULER MODULE DESCRIPTIONS

The modules described in this section are subordinate to the Radar Scheduler process. A functional description of each of the modules, and any subroutines subordinate to them is given below.

1. **Initialization**

The function of the initialization module is to create and initialize any data structures, local to the Radar Scheduler process, that must be shared between the Radar Scheduler modules. The creation of these data structures is necessary only for access type data structures which form linked lists. Allocation of memory for these data structures prior to the execution of the Radar Scheduler process serves to increase the efficiency of the main scheduling loop by eliminating the need to allocate memory for each new node during scheduling. This module is executed only one time, when the Radar Scheduler process is loaded for execution.

The decision to place these variables in a separate module instead of the package specification for the Radar Scheduler process is based on the principle of information hiding. Since these variables need to be shared by other Radar Scheduler modules they must be in a package specification. If they were placed in the Radar Scheduler's package specification, then modules other than Radar Scheduler modules would be able to access them.

The Initialization module does not consume any common memory interface data structures. It does produce the initial pool of Output Channel Control Buffer nodes and a pool of Radar Event Control Table nodes.

The following data structures are declared in module specification and are local to the Radar Scheduler process:

- **rdrinit** is a constant representing the maximum amount of time which can be spent in the beam selection mode.
- **srch_que** is a constant assigned to the event type search.
- **trk_que** is a constant assigned to the event type track.
- **sr_que** is a constant assigned to the event type special request.
- **mssl_que** is a constant assigned to the event type missile.
- **srch_dwls** is a constant representing the maximum number of search events that can be scheduled in one interval.
- **trk_dwls** is a constant representing the maximum number of track events that can be scheduled in one interval.
- **sr_dwls** is a constant representing the maximum number of special request events that can be scheduled in one interval.
- **mssl_dwls** is a constant representing the maximum number of missile events that can be scheduled in one interval.
- **srch_pcnt** is a constant representing the percentage of radar resources allotted to each search dwell.
- **trk_pcnt** is a constant representing the percentage of radar resources allotted to each track dwell.

- **sr_pcnt** is a constant representing the percentage of radar resources allotted to each special request dwell.
- **mssl_pcnt** is a constant representing the percentage of radar resources allotted to each missile dwell.

The following data structures are used to control program execution:

- **ppl** is an index used to control traversal of the priority event list.
- **intvl_num** is used to keep track of the number of intervals that have been executed.
- **sp** is a working pointer for traversing a linked list of search nodes.
- **tp** is a working pointer for traversing a linked list of track nodes.
- r is a working pointer for traversing the Radar Event Control Table.
- **rect_ptr** is a pointer which always identifies the first node in the Radar Event Control Table.
- **pool_ptr** is a pointer which always identifies the first node in the pool of available Radar Event Control Table nodes.
- **buff_ptr** always points to the current Channel I/O Control node.
- **cm_ptr** always points to the current Radar Output node.

The Radar Event Control Table is the primary data structure used by the Radar Scheduler process to schedule the mix of radar events for each interval. This data structure contains five major fields:

- **srch_dwl** is a mirror image of the search node data structure (see Section C.1.a of this chapter).
- **trk_dwl** is a mirror image of the track node data structure (see Section C.1.b of this chapter).
- **ocb_data** contains the information that is transmitted to the external dwell buffers. An explanation of the data fields in this record is is given in Sections C.2.d and e of this chapter.
- **beamid** corresponds to the beam identifier used by the **srch_dwl** or **trk_dwl** "**bid**" data field and is used for testing the logic of the Radar Scheduler algorithm.
- **dru** holds the total percentage of radar resources consumed after the selection of the current beam.
- **nxt_event** is a link to the next node in the linked list that makes up the Radar Event Control Table.

There are no data structures locally declared within this module body. The procedures **make_pool** and **ex_buff_create** are declared local to the module body and are used for the initialization process. An algorithmic description of the Initialization module is given in Figure 4.2 .

a. Make Pool

The procedure Make Pool is derived from the PL/I-80 version of RSM1A. This procedure is declared within the body of the Initialization module described

```
Begin Radar Scheduler Initialization;
    Create a pool of Output Channel Control Buffer nodes;
    Create a pool of Radar Output Interface nodes;
    Create a pool of Radar Event Control Table nodes;
    Initialize working pointers for Radar Scheduler;
End Radar Scheduler Initialization;
```

Figure 4.2 Initialization Module Algorithm.

above. The purpose of this procedure is to make a pool of Radar Event Control Table nodes.

The procedure declares two local pointers, "p" and "q" for creating a linked list of Radar Event Control Table nodes. The variable **numb_nodes** stands for the number of nodes to be created.

b. Create Dwell Buffer Pools

The procedure Create Dwell Buffer Pools is derived from the PL/I-80 version of RSM1B. The task of this procedure is to create two circular linked lists for the Output Channel Control Buffer and two circular linked lists for the Radar Output Interface nodes.

The procedure declares the following working pointers locally for the purpose of creating the circularly linked lists:

- **p1** and **q1** are Output Channel Control Buffer pointers.
- **p2** and **q2** are Radar Output Interface pointers.

The following data structures are used for execution control flow within the procedure:

- **length** is the length of the circular linked lists being created.
- **ctr** is a variable used to keep track of the number of nodes being created.
- **i** is an index to control the number of circular linked lists being created.

2. **Swap Dwell Buffers**

Functionally, the Swap module manages the Radar Scheduler's access to the two external dwell buffers described above. The Swap module must insure that a minimum number of available nodes are present in the pool for consumption by the

Radar Scheduler during each scheduling interval. The Swap module is called from the beginning of the main control loop in the Radar Scheduler process.

The Swap module consumes the global pointer variables to the common memory interface pools of Output Channel Control Buffer and Radar Output nodes. This module does not produce any common memory interface data structures. The algorithm for this procedure is given in Figure 4.3

The input/output parameters are the only data structures declared locally and are passed by reference. These data structures are:

- **buff** is a pointer to the Output Channel Control Buffer.
- **cm** is a pointer to the common memory interface Radar Output buffer.
- **index** is the index to the current buffers.

```
Begin Swap Dwell Buffers:
    If the index is two then
        the index gets one:
    else
        the index gets two:
    end if;
    change dwell buffers;
End Swap Dwell Buffers;
```

Figure 4.3 Swap Dwell Buffers Algorithm.

3. **Radar Event Priority Enhancement**

The function of this module is to ensure that the mix of radar events being considered for selection is optimized by their respective priorities in the Radar Event Priority List. The design is based on Digital Equipment's VMS Operating System Priority Enhancement Algorithm. Radar events having base priorities lower than the enhancement value have the capacity for dynamic prioritization. The procedure examines each event to see if its priority can be enhanced. If the event's priority needs to be enhanced, its priority value is increased. Execution of this module takes place prior to the Beam selection and Synchronization. When executed, the module produces a reprioritized Radar Event Priority List. [Ref. 5: p. 76]

```
Begin Priority Enhancement;
    Reset the last time executed for all radar events;
    Traverse the enhanceable portion of the priority list;
        If the event is enhanceable and its queue is not
        empty then
            If the time between scheduling of the event is
            greater than the allowed time then
                If the event's current priority is above
                the standard enhancement value but below
                the lowest enhancement value then
                    Increment the event's priority by 1;
                Else
                    Increment the event's priority by 4 but
                    not above the low enhancement value;
                End Enhance;
                Reorder the priority event list;
            End Last time executed;
    End Traverse the priority list;
End Priority Enhancement;
```

Figure 4.4 Priority Enhancement Algorithm.

The enhancement module does not consume any common memory interface data structures. The input parameter to this procedure is the variable **elapsed_time** which is passed by value from the Radar Scheduler process. The parameter **elapsed_time** is used to update the Radar Event Priority List **ltx** data field. The algorithm for the Enhancement module is presented in Figure 4.4

This module uses the procedure **ripl** to remove and insert a node from its present position in the priority list to its new position in the list. The procedure is declared locally in the Enhancement modules body.

a. Remove and Insert

The Remove and Insert procedure is part of the Enhancement module. The procedure removes a node from its current position in the priority list, inserts the node at its new priority and then resets the current priority values for the other events.

The following data structures are declared locally:

- **curnt** is an input parameter passed by value representing the event's current priority in the list.
- **new_p** is an input parameter passed by value representing the event's new priority in the list.
- **p** is an index used to traverse the priority list.
- **b4** is a variable that holds the last value of the index "p".
- **temp1** and **temp2** are temporary storage variables for the indexes.

4. **Radar and Computer Synchronization**

The function of this module is to ensure that the Radar Control Group time is synchronized to the radar time. The scheduler waits for a clock update from the radar. According to AEGIS performance specifications, the scheduler must schedule dummy dwells, if the update is greater than two seconds. The Radar Scheduler design calls for the process to be "blocked" until synchronization has been accomplished. The scheduler is made "ready" upon synchronization of clocks and Track File time updates. The Radar Scheduler must alert the RSC (or in our case the Switch Action and Display process) to the pending update. This requirement is not implemented in this version of the scheduler model. For clock updates of less than two seconds, immediate synchronization is carried out and the the current scheduling interval is modified to conform with the new Radar Control Group time. [Ref. 5: p. 78]

```
Begin Synchronization;
    If I_to_R.resynch_time minus Radar Scheduler time is
    greater than 2 seconds then
        Wait until the Radar Scheduler time equals the
        resynch_time;
    Else if the resynch_time minus Radar Scheduler time is
    greater than or equal to zero but less than or equal to
    2 seconds then
        Radar Scheduler time gets radar time;
End Synchronization;
```

Figure 4.5 Synchronization Algorithm.

This module consumes the synchronization variable within the Radar Return (input) Interface data structure. It does not produce any common memory interface

data. Real time data is produced for use by the Radar Scheduler when a clock update occurs.

The modules local data structures have not been designed. The algorithm for Radar and Computer Synchronization is given in Figure 4.5 .

5. **Beam Selection Routines**

This module contains procedures used in the Beam Selection process of the Radar Scheduler. The code for the Beam Selection process is actually part of the Radar Scheduler code and the functional description for Beam Selection is given there. This module is a package containing two procedures and a function. Their functional descriptions are given in the following subsections of this section.

This module contains one local data structure, a working pointer **p** which is used by the routines in this module's body. The pointer is declared outside of the routines inorder to increase their efficiency and prevent heap or stack over flow which would occur by repeated calls to these procedures/function.

This module does not consume or produce any common memory data structures.

a. Add Linked List

Functionally, the procedure Ilend is used to insert a Radar Event Control Table node at the end of a linked list. This procedure is derived form the PL/I-80 module RSM3A. The procedure declaration is contained in the Beam Selection module described above.

The parameters to this procedure are pointers, **q** and **s**, which are passed by reference. The pointer **q** points to the head of the list. The pointer s points to the node which is to be inserted at the end of the linked list. The pointers consumed in this procedure are Radar Event Control Table data structures local to the Radar Scheduler modules only. There are no common memory interface data structures consumed by this procedure.

There are no local data structures produced by this procedure. The working pointer **p** is declared in the body of the Beam Selection Routines module.

The algorithmic description for this procedure is given in Figure 4.6 .

b. Get RECT Node

The function Get RECT Node locates the first available Radar Event Control Table node from the pool and returns a pointer to that node. The function also resets the head of pool's linked list to the next available node.

```
Begin llend;
    If first list q is null then
        list q gets node s;
    Else
        Find end of list q;
        End of list q gets node s;
End llend;
```

Figure 4.6 llend Algorithm.

There are no common memory interface data structures consumed by this procedure.

There are no local data structures produced by this procedure. The working pointer p is declared in the body of the Beam Selection Routines module.

The algorithmic description of this function is given in Figure 4.7 .

```
Begin Get RECT Node:
    If pool is empty  then
        Create a new pool pointer;
    End if;
    p gets pool pointer;
    pool pointer gets next node in linked list;
    Return pointer p;
End Get RECT Node;
```

Figure 4.7 Get RECT Node Algorithm.

c. Free RECT Node

Functionally, this procedure returns an unused Radar Event Control Table node to the pool of available nodes. The procedure Free RECT node together with the function Get RECT Node are used to manage pointer storage.

There are no common memory interface data structures consumed by this procedure.

There are no local data structures produced by this procedure. The working pointer **p** is declared in the body of the Beam Selection Routines module.

Figure 4.8 gives the algorithmic description of the procedure Free RECT Node.

```
Begin Free RECT Node:
    Set node q's link to null;
    If pool is empty then
        pool pointer gets node q;
    Else
        Insert node q at end of pool list;
    End if;
End Free RECT Node;
```

Figure 4.8 Free RECT Node Algorithm.

6. **Supplementary Dwell Processing**

The function of the Supplementary Dwell Processing module is to do sufficient processing on each selected beam to ensure enough dwell information is generated for correct transmission by the radar. The minimum information required is:

- the scheduled dwell's transmission time,
- the dwell index number,
- the stable space bearing and elevation beam position,
- the radar end the dwell is to be transmitted from and,
- the radar transmission parameters.

The data structures used by this module have not been decided upon [Ref. 5: p. 82].

The algorithm for this module has not been written.

7. **Dwell Array Face Assignment**

The function of this module is to assign an array face to each selected beam. The NPS model assumes that the ship is not underway and on a heading of 000

degrees true, therefore the model does not emulate gyro inputs. Hence, stable space coordinates equate to ship's deck space coordinates. The result of this is that the transformation matrix and array face bearing limits generated by the Beam Stabilization process are constant. The Assignment module is used to select the array face for the beam's transmission by comparing the beam's bearing to the limits found in the array face bearing limits table. This assignment consumes the common memory interface data for array face bearing limits. No common memory data is produced. [Ref. 5: pp. 82-83]

No internal data is consumed by this module and there are no locally declared data structures in this module. It does produce the face assignment data for the selected beam which is located in the Radar Event Control Table data structure.

This module is not implemented in the Radar Scheduler model. An algorithmic description of the module is given in Figure 4.9 .

```
Begin Dwell Array Face Assignment;
    If beam bearing is between first limits then
        Beam face assignment gets one;
    Else if beam bearing is between second limits then
        Beam face assignment gets two;
    Else if beam bearing is between third limits then
        Beam face assignment gets three;
    Else
        Beam face assignment gets four;
End Dwell Array Face Assignment;
```

Figure 4.9 Dwell Array Face Assignment.

8. **Comply With Radar Doctrine**

The function of this module is to ensure that the selected beam's transmission parameters comply with the doctrines imposed by the program operators.

The common memory data consumed by this module includes the radar silence flag, produced by the Command and Decision User services process, and three

doctrine commands produced by the Switch Action and Display process. The doctrine commands consumed are:

- the radar transmitter power flag,
- the radar power option flag and,
- the inhibited radiation regions.

This module does not consume any internal data and there are no locally declared data structures. [Ref. 5: p. 84]

This module does produce data for the Output Channel Control Buffer data structure of the Radar Event Control Table.

This module is not implemented in this program. The algorithm for this module is presented in Figure 4.10 .

```
Begin Comply With Radar Doctrine:
    If the radar silence flag is false then
        Set the transmitter power based on the flags set
        by the Switch Action and Display process;
        If beam's lie within the inhibited radiation
        regions then
            Set transmitter flag to false;
        End if;
    Else
        Set transmitter flag to false;
    End if else;
End Comply With Radar Doctrine;
```

Figure 4.10 Comply With Radar Doctrine Algorithm.

9. **Satisfy SPY-1A Hardware Constraints**

The function of this module is to optimize the available radar resources for dwell transmission during each scheduling interval. The AEGIS Program Specifications call for the use of digital filters to optimize dwell transmission parameters. In the NPS model, a fixed resource percentage scheme is used to calculate the amount of radar resources consumed. Each radar event is assigned a constant

resource percentage requirement. As the requested beams are selected for scheduleding, their constant resource requirements are subtracted form the total radar resources available. The total radar resources for the start of each scheduling interval is 100%. Beams are no longer selected once the available resources have been depleted. filenum rsm9

```
Begin Satisfy Hardware Constraints:
    Ascertain the beams identity and set
    percent equal to beams alloted resource:
    Radar resources used gets radar resources plus percent;
    If radar resources used is less than 100% then
        set dwell resources used to 100% minus the
        radar resources used:
        Set the hardware constraints flag to true:
    Else
        Set radar resources used to radar resources used
        minus percent:
        Set the hardware constraints flag to false:
    End if else:
End Satisfy Hardware Constraints;
```

Figure 4.11 Satisfy Hardware Constraints Algorithm.

This module does not consume or produce any common memory data structures.

The input parameters to the Satisfy SPY-1A Hardware Constraints procedure are as follows:

- **id** is a que type identifier used to determine what percentage of resources are required. This parameter is passed by value.
- **rru** is an input/output parameter containing the running total of radar resources used for the current scheduling interval. The parameter is passed by reference.
- **dru** is an input/output parameter containing the total percentage of radar resources consumed after selection of the current beam.
- **hwc** is a boolean output parameter which is set to true if the hardware constraints are satisfied. The parameter is passed by reference.

The following data structures are locally declared:

- **srch_pcnt** is a constant representing the percentage of resources allotted for each search dwell.
- **sr_pcnt** is a constant representing the percentage of resources allotted for each special request dwell.
- **trk_pcnt** is a constant representing the percentage of resources allotted for each track dwell.
- **mssl_pcnt** is a constant representing the percentage of resources allotted for each missile dwell.
- **percent** is a temporary variable used to hold the percent of resources being considered.

The algorithmic description for the Satisfy Hardware Constraints procedure is presented in Figure 4.11 .

10. **Place Dwells Into Dwell Buffers**

The function of this module is to transmit the formatted dwell information to the two external dwell buffers. Formatting is complete upon satisfying the hardware constraints. If the selected beam satisfies those constraints, then it is transmitted to the external dwell buffers for radar transmission.

```
Begin Fill External Dwells;
    Point to the current Output Control Channel Buffer;
    Set the Output Control Channel Buffer data structure to
    the ocb_data structure of the Radar Event Control Table;
    Point to the Current R_to_O data structure;
    Set the R_to_O data structure to the appropriate data
    fields in the Radar Event Control Table;
End Fill External Dwells;
```

Figure 4.12 Fill External Dwells.

This procedure consumes the current pointers to the external dwell buffers which are passed by reference in the following input/output parameters:

- **p1** and **p2** are pointers to the Output Control Channel Buffer common memory data structure.
- **p3** and **p4** are pointers to the R_to_O common memory data structure.
- **pr** is a pointer to the Output_Control Channel Buffer data contained in the internal Radar Event Control Table data structure for the current the current selected beam.

There are no other locally declared data structures in this module. Figure 4.12 gives the algorithmic description of this procedure.

11. **Radar Load Evaluation**

The function of this module is to maintain a running total of the amount of radar time being expended in tracking targets. The total is reset to zero each time the Load Evaluation process tells the Radar Scheduler to reset its' track time counter. Beam transmission data is then sent to the Load Evaluation process. This module is executed only if the selected beam has satisfied the hardware constraints. [Ref. 5: p. 87]

The Load Evaluation module consumes the common memory interface from the load evaluation process (TAB9.LIB). The data structure consumed is the track time counter reset flag. The common memory interface information produced by this module includes information concerning the transmitter duty factor for the current beam, the total time spent on dummy dwells during the interval, and the number of horizon and above horizon search beams that were not scheduled during the interval.This information is placed in the common memory interface of TAB65.LIB. [Ref. 5: p. 88]

Internal data consumed by this module includes the beams identity and the percentage of resources used by the beam. No internal data is produced by this module. Also, this module does not use any locally declared data.

The algorithm for Load Evaluation is given in Figure 4.13 .

12. **Elapsed Time This Interval**

The function of this module is to compute the amount of time spent scheduling and formatting the radar beams during the interval.

No common memory interface data is consumed or produced by this module.

The module does consume the internal Radar Scheduler data structure **rs_time**. Each time the module is executed a new value for the elapsed time is produced.

The Elapsed Time This Interval module uses the following local data structures:

- **et** is an input/output parameter that holds the elapsed time.
- **rtim** is an input/output parameter that holds the Radar Scheduler time.
- **oltim** is a variable to hold the old time value.

Figure 4.14 shows the algorithm used by the Elapsed Time procedure.

```
Begin Radar Load Evaluation;
    Increment the Duty factor variable (fore or aft);
    Increment the phase shifter variable (fore or aft);
    If the track time counter reset variable is true then
        Set track time counter to zero;
    If selected beam is a track or missile beam then
        Increment the track time counter;
    If the selected beam is a dummy dwell then
        Increment the dummy dwell time variable;
    If the selected beam is a search beam then
        decrement the unsked search beam variable;
End Radar Load Evaluation;
```

Figure 4.13 Radar Load Evaluation Algorithm.

```
Begin Elapsed Time;
    Old time gets the value of the real time;
    Real time gets the current value of real time;
    Elapsed time gets elapsed time plus real time
    minus the old time;
    Return the new values of elapsed time and real time.
End Elapsed Time;
```

Figure 4.14 Elapsed Time Algorithm.

13. **Radar Event Control Table Analysis**

This module is designed to dump selected fields from the Radar Event Priority List and the Radar Event Control Table into a file called RSOUT.TXT. When the Radar Scheduler program completes its execution, the file will contain the requested beams from the Radar Event Priority List and the selected beams from the Radar

Event Control Table. The results can be analyzed by comparing the requested beam data to the scheduled beam data.

```
Begin Radar Scheduler Dump;
    Traverse the Priority Event List;
        If Priority Event List status is true then
            If search or special request beam queue then
                Traverse search or special request beam queue;
                    Display beam identifier;
                    increment beam position count;
                End traverse search or special request beam queue;
            Else
                Traverse track or missile beam queue;
                    Display beam identifier;
                    Increment beam position count;
                End traverse track or missile beam queue;
            End if else;
            Start new line and display beam positions;
        Else
            Display "no requests this interval";
        End if else;
    End traverse Priority Event List;
    Traverse Radar Event Control Table;
        Display beam identifiers;
        Display dwell number;
        Display resources;
    End traverse Radar Event Control Table;
End Radar Scheduler Dump;
```

Figure 4.15 Radar Schedular Dump Algorithm.

This module consumes the Radar Event Priority List common memory data structure. It does not produce any common memory data.

This module also consumes the Radar Event Control Table internal data structure. It does not produce any internal data.

The local variables used in this module are defined below:

- **dsh** is a string constant containing a dashed line for formating the output.
- **sptr** is a working pointer used in traversing the Radar Event Priority List's search or special request beam queue.
- **tptr** is a working pointer used in traversing the Radar Event Priority List's track or missile beam queue.
- **p** is a working pointer used in traversing the Radar Event Control Table.
- Control Table.

The dump procedure declared in this module uses the pointers described above. The procedure also defines the following local data structures:

- **ctr** is used as an index for traversing the Radar Event Priority Event list.
- **start** is a variable used to keep track of the starting value used in the "FOR" loop control structure.
- **ctn** is a variable used to keep track of the queue position.

The the algorithm for this procedure is presented in Figure 4.15 .

14. Free Radar Event Control Table Memory

The function of this module is to return the Radar Event Control Table nodes to the pool of available nodes at the end of each scheduling interval so they can be reused. This is accomplished by linking the Radar Event Control Table to the End of the node pool. Then as the nodes are required, the Get RECT Node function from the RSM5 module returns them to the Radar Event Control Table.

```
Begin Free Memory;
    Find the end of the RECT node pool;
    Insert the Radar Event Control Table
    at the end of the RECT node pool;
    Return null Radar Event Control Table pointer;
End Free Memory;
```

Figure 4.16 Free Memory Algorithm.

The Free Memory module does not produce or consume any common memory data structures.

The module consumes the internal data structure, Radar Event Control Table, and reproduces the pool List.

The module produces the local pointer **p** for use by the Free Memory procedure. Figure 4.16 contains the algorithmic description of this procedure.

F. RADAR SCHEDULER COMMON SERVICE ROUTINES

The Common service routines used are designed to be accessible to all the major processes. For simplicity, the common service routines used by the Radar Scheduler program have been included in the Global module. Some of the previously designed Common Service Routines' functions have been replaced by standard Ada packages, such as the type conversion and string manipulation functions. The procedures await, advance, ticket etc., are expected to be replaced by MCORTEX procedures and appear only as stubs in this program. The design decisions for the Common Service Routines implemented in this model are included here.

1. Random Number Generator

The Random Number Generator is a function that returns a pseudo-random number. The function uses a congruence algorithm to generate numbers in the range from zero to the "t". In this case "t" is 31099.

```
Begin Random Number Generator;
    If this is the first call to this module then;
        set x equal to seed number;
    Compute the random number;
    set "x" equal to the random number;
    Return the value of "x" to the caller;
End Random Number Generator;
```

Figure 4.17 Random Number Generator Algorithm.

This routine does not consume or produce any common memory or Radar Scheduler data structures. The module does consume the variable "x" which is declared in the Global package body along with the function. Unlike the function, the variable "x" is hidden from the users of the Random Number Generator. This variable is

initialized by the Global package body and thereafter set to the pseudo-random number returned after each execution of the function. The variable "x" is then used as a seed for each successive call to the function.

The data structures defined by this function are:

- **a** is a constant approximately equal to the square root of "t".
- **b** is a constant that is relatively prime to "t".
- **t** is a constant which defines the upper bound of the numbers generated.
- y is a temporary variable used to calculate the random number.

Figure 4.17 presents the algorithm for the random function.

2. Clock Routine

This Common Service Routine is designed to simulate a real time millisecond clock. Each time this function is called the variable **time** is incremented. This new value of time is then returned to the calling process.

This module does not consume or produce any common or internal data structures.

The only data structure used by this function is the variable **time**, which is declared in the package body of the Global module. The Clock Algorithm is shown in Figure 4.18 .

```
Begin Clock;
    Increment "time";
    Return the value of "time";
End Clock;
```

Figure 4.18 Clock Algorithm.

3. **Initialize Radar Event Priority List**

The purpose of this procedure is to initialize the data fields in the Radar Event Priority List. It is designated as a Common Service Routine because it must be executed prior to any process using the list.

This routine does not consume any common memory data structures.

No Radar Scheduler internal data structures are produced or consumed by this routine.

There is only one local variable declared by this procedure, the variable "i", which is used as an index for traversing the Radar Event Priority List.

The Initialization algorithm is presented in Figure 4.19 .

```
Begin Initialization;
    For "i" gets one to maximum number of events loop;
        Set initial values for the common data fields;
        Assign allowed execution periods based on
        the index "i" and the description given in
        Table &prilst;
        Assign beam queue types maximum nodes based on
        the index "i" and the description given in
        Table &prilst;
        Set the link to the next event in the list;
    End loop;
    Reset last link to zero;
    Assign event names inaccordance with Table &prilst;
End Initialization;
```

Figure 4.19 Priority Event List Initialization Algorithm.

V. TEST PLAN

The testing requirements for the NPS model of the Radar Scheduler process have a limited set of goals. The primary goal is to test for logical correctness and secondarily to test the performance of the code generated by JANUS/ADA's compiler.

Testing for logical correctness entails verifying that the program conforms to its specifications, implements the design algorithm, and produces the required output. Testing of the Radar Scheduler process for this thesis was done primarily by a "bottom up", "white box" approach. The reason for taking this approach as opposed to a top down approach is that the basic design had already been implemented once in the PL/I-80 version of the code. Since this was the case, it was felt that the overall design was sound and there was very little risk in implementing and testing the modules from the bottom up. Furthermore, this allowed for more extensive testing, as there was little need for module stubs.

A test harness was designed for each module with an effort towards exercising, as much as possible, each branch of the code for correct execution and robustness. The subordinate modules were tested first. The test harnesses were then expanded to incorporate the modules at the next higher level. At each phase of the testing the test harnesses were modified to allow the input of relevant data to exercise the branches of the code and then record the results for examination. In some cases print statements were inserted into the code to ensure that a particular branch was being executed properly.

A. DESIGN AND DOCUMENTATION OF TEST MODULES

1. Search Management Test Harness

The Search Management test harness is made up of two modules, the Search Management Initialization module and the Fill Search Queue module. Collectively these two modules provide the search and special request beams for the Radar Scheduler Process.

The Search Management Initialization module is executed once. Its' purpose is to allocate memory for search nodes (TAB1.LIB), and create empty request queues for each search and special request event in the Radar Event Priority List (TAB0.LIB).

The Fill Search Queue module is responsible for filling the search and special request queues with the proper beam data. The data supplied by this module includes the beam mode, a unique beam identifier, beam position, instrumented range, blanking gates, end of frame indicator, and a requestor identity. This design version imposes a static set of beam requests, for test purposes. Whenever the module is executed, the same set of beam requests are placed in the queues. The static set of beam requests are used to facilitate the data analysis of the Radar Scheduler and is not meant to accurately model the Search Management process. The source code for the Search Management modules is contained in Appendix D.

2. **Track Management Test Harness**

The Track Management test harness is composed of the Track Management module (TRCM) and a Track Management Initialization module (TMM1). The Initialization module is designed to allocate memory for track nodes (TAB2.LIB) and create empty request queues for each track and missile event in the Radar Event Priority List (TAB0.LIB)

The main Track Management module is used to search through the Track File correlating track modes to radar track event modes. When a correlation is found, the track is added to the Priority Event List's request queue. This module does not accurately model the Track Management process and again is used only for testing the Radar Scheduler Source code. The source code for these modules is located in Appendix D.

3. **Detection Processing Initialization Module**

The Detection Processing Initialization module (DPM) creates the Track File and initializes its data fields with viable data for testing the Radar Scheduler. The number of Track File nodes initialized is determined by the test harness operator at run time, when he is directed to provide the "number of tracks" to be initialized. Calculation of the data field values in the track node are based on the values returned by the pseudo-random number generator. Therefore successive runs of the test harness with identical input from the operator will generate the same Track File data each time. The source code for this module is located in Appendix D.

4. **Operator Interface Module**

The Operator Interface module provides the user of the test harness the ability to alter the number of tracks to be initialized, the desired number of intervals to be run, and how often the results are to be displayed. Actually the results are sent to a file

that can be examined by the user after the test run. A short messages is printed to the screen whenever results of the scheduling interval are sent to the file. This can also be used to time the scheduling interval loop without the overhead of the routines that are only executed once to initialize the data structures.

The operator is cautioned that using a very large number for the number of intervals and a small interval display delay period will generate a very large file of test results and may cause a heap or stack overflow. If the operator desires such a run, then the print functions in RSM13.PKG can be easily modified to print to the screen, by eliminating the word "Text" from the print commands.

B. DATA ANALYSIS METHODS

The Radar Scheduler model was designed so that the results of any particular scheduling interval may be recorded in the file RSOUT.TXT. For each scheduling interval being recorded, the output file holds a section of information on the requested beams from the Radar Event Priority List and a section of information on dwells scheduled from the Radar Event Control Table.

The requested beam section shows the scheduling interval number and lists the events in their current priority order. Along with each event is a list of the requested beams if any, and their position in the request queue. All the beams are identified by a unique beam identifier.

The scheduled dwell section shows the interval number the dwells were scheduled in and a list of all dwells scheduled. For each dwell the following is provided:

- the unique beam identity of the dwell,
- the percentage of resources left after formatting the selected beam into a dwell, and
- the dwell index number of the selected beam.

The Radar Scheduler's output is analyzed by comparing the requested beam identities to the selected beam identites of the scheduled dwells. The results recorded over several intervals indicate how well the model is optimizing the radar resources and if the radar events are being scheduled efficiently.

C. TIMING ANALYSIS

To test the timing of Radar Scheduler program, a short message is sent to the screen each interval display delay period. When the first message is displayed, the Radar Scheduler has completed its initialization of data structures and completed the

number of scheduling intervals equal to the display delay period. In order to avoid the initialization overhead at the start of the program, timing must start when the first message appears and not when the program is first executed. Then the operator may end timing on any later interval display delay period. This way the operator can get a good approximation of the time it takes to complete one scheduling interval.

In order to calculate the average scheduling interval time, the overhead from the terminal display and writing to output file must be accounted for. To account for this, the assumption is made that the time to print to the screen and dump the results to the output file is approximately equal each time it occurs. In other words, the interval display procedure takes approximately the same amount of time each time it is executed.

Let "Tr" equal the total time recorded for the test run. Let "Td" equal the time it takes for one execution of the display procedure. Let "Ti" equal the execution time for one scheduling interval. To calculate "Ti" the following test procedure was used:

- Three runs of the Radar Scheduler were made using the same input. The results were recorded and their average was used as "Tr".
  a. The interval display delay period was set on 500.
  b. The number of intervals was set to 1000.
  c. Timing was started on the first interval display delay period, 500.
  d. Timing was stopped on the last interval display delay period, 1000, which caused only the last display to be included in the total time recorded.

- Three runs of the Radar Scheduler were taken again using a new interval display delay period. The the run times were recorded and again the average was used as "Tr".
  a. The interval display delay period was set on 100.
  b. The number of intervals was set to 1000.
  c. Timing was started on the fifth interval display delay period, 500.
  d. Timing was stopped on the last interval display delay period, 1000, which caused five display times to be included in the total time recorded.

For the first set of runs the following equation holds:

$$Tr1 = 500Ti + Td.$$

The second set of runs uses the formula:

$$Tr2 = 500Ti + 5Td.$$

Solving the two simultaneous equations for Ti yields:

$$Ti = (5Tr1-Tr2)/2000.$$

Test runs were made of the code generated by JANUS/ADA compiler with all the standard pragmas, and then again on the code generated when the pragmas were turned off. The pragmas that were turned on and off during compilation are shown at the beginning of each source code module. For consistancy, all the modules include the same pragma statements. The pragmas that were turned on and off are: **arithcheck, rangecheck, debug,** and **enumtab.**

The arithmetic check pragma controls the generation of arithmetic overflow checks. Code is generated during compilation to check all mathematical expressions when this pragma is turned on. [Ref. 6: p. B-1]

The range check pragma controls the generation of range checks for subrange variables and array subscripts. The range check code is generated at compile time when the pragma is turned on. [Ref. 6: p. B-2]

The debug pragma controls the generation of line number and procedure names. This information is used by the walkback which is generated after a run-time error. The code generated when this pragma is on would only affect performance when a run-time error occurs. [Ref. 6: p. B-1]

The enumeration table pragma controls the generation of enumeration tables. These tables are only necessary when the programer is doing I/O with enumeration types. The generation of these tables at compile time does not affect performance, but it does use a lot of storage space. [Ref. 6: p. B-1]

Since the above pragmas are turned on by default, they have been explicitly turned of in the code. They can be turned back on at compile time by the conditional compilation feature. When this feature is off, as it is by default, the lines preceded by a "@" are treated as comments. If the conditional compilation is turned on with the "c" command at compile time, then the lines preceded by the "@" symbol are compiled. [Ref. 6: p. B-1]

The first executable code was generated with the pragmas on and the second was generated with the pragmas off. The test results are shown in Table 4 .

Run-time test results of the code produced by the JANUS/ADA compiler showed almost a two to one increase in speed when it was recompiled with the arithmetic and range checking features turned off. This shows that although there was a significant increase in speed, it was not without sacrificing some of features that the Ada language includes. Even with the pragmas turned off, the best speed of 213.4 milliseconds is still ten times the median scheduling interval time of 21 milliseconds required.

TABLE 4

PERFORMANCE RESULTS

PRAGMAS ON

```
NUMBER OF TRACKS: ..................... 50
NUMBER OF INTERVALS: .................. 1000
INTERVAL DISPLAY DELAY: ............... 500
NUMBER OF INTERVALS TIMED: ............ 500
AVERAGE RUN TIME Tr1: ................. 228.58 sec

NUMBER OF TRACKS: ..................... 50
NUMBER OF INTERVALS: .................. 1000
INTERVAL DISPLAY DELAY: ............... 100
AVERAGE RUN TIME Tr2: ................. 231.48 sec

TIME PER SCHEDULING INTERVAL "Ti": ... 0.4557 sec/int
SCHEDULING RATE: ...................... 2.2 int/sec
```

PRAGMAS OFF

```
NUMBER OF TRACKS: ..................... 50
NUMBER OF INTERVALS: .................. 1000
INTERVAL DISPLAY DELAY: ............... 500
NUMBER OF INTERVALS TIMED: ............ 500
AVERAGE RUN TIME Tr1: ................. 107.49 sec

NUMBER OF TRACKS: ..................... 50
NUMBER OF INTERVALS: .................. 1000
INTERVAL DISPLAY DELAY: ............... 100
AVERAGE RUN TIME Tr2: ................. 110.65 sec

TIME PER SCHEDULING INTERVAL "Ti": ... 0.2134 sec/int
SCHEDULING RATE: ...................... 4.6 int/sec
```

The tests were run on the Z-100's 8-Bit microprocessor. The target processor is the iSBC 86/12A Single Board Computer with a 16-Bit microprocessor, which will increase the performance. However, it is doubtfull that this will be enough to over come a factor of ten.

VI. CONCLUSIONS AND RECOMMENDATIONS

This thesis has developed a JANUS/ADA model of the Radar Scheduler process. The model is a first effort in implementing one of the AEGIS AN/SPY-1A Radar Control Group modules in JANUS/ADA and as such it is a part of the continuing research effort at the Naval Postgraduate School to test the feasibility of replacing the AEGIS main frame computer suit with a multi-microprocessor system. Additionally, the Radar Scheduler provides an example for studying the performance of time critical programs written in ADA.

The design of the Radar Scheduler model incorporates 14 modules. The modules were designed to be integrated into the overall model with a minium of changes required. In support of this, Chapter IV, serves as the Radar Scheduler Design Document to be used as a reference for the future implementation and integration of the Radar Scheduler Process with the remaining Radar Control Group modules. As such, any changes to the Radar Scheduler's design should be reflected in the design document.

Logical testing of the individual modules was increasingly more dificult as they were integrated into the main program. There were several reasons for this. First, the number of logical paths to be examined grew very rapidly as the modules were integrated. In some cases, the lower modules had tested satisfactorily, isolated in their own test harness. Then later, while acting in conjunction with the higher level modules, they developed problems which were not accounted for by the test harness. For example, the dynamic allocation of local access data structures, and some recursively designed procedures, functioned well until they were stressed by the load of the Radar Scheduler test harness. Under this stress, stack overflow became a problem. Declaring the access types outside the procedures in the package body, and making the recursive procedures iterative, solved the problem.

Determining what the best testing criteria is, and formulating a testing strategy is difficult without a more rigorously defined specification. The logical correctness of the module can not be determined from statements such as "in a timely manner". However, the Radar Scheduler output indicates that the requested beams are being scheduled in a priority order, and that those beams that are not scheduled during the

interval are being reprioritized and scheduled in later intervals. The radar resources remaining at the end of the interval indicate that the resources are being consumed until there is no longer enough to fill the remaining requests for that interval.

Two conclusion are evident from the preliminary real-time testing. First, the code generated with the range and arithmetic checking turned off executes at twice the speed of the code normally generated by the compiler. Second, the best run-time was ten times slower than the median scheduling interval time required. In view of this, further testing should be done to isolate the cause. The most time intensive portions of the program should be analyzed first. Identification should be made by measuring the execution time for the most likely modules and weighting the results by the number of times the module is executed by the Radar Scheduler process. Once the modules have been identified, the actual cause of the time delay can be asessed. The algorithms and data structures can be examined for improvement. This way, a more accurate accounting of the module design, and the feasibility of real-time programming in ADA can be made.

The supplementary processes (TRCM, SRCM, DRCM, and ARCM) were developed as test harnesses to supply the Radar Scheduler with radar event requests. These processes will be fully implemented in the Janus/Ada programming language as the NPS AEGIS Project progresses. Ultimately the processes will be integrated into a multi-microprocessor model of the Radar Controller.

To completely model the Radar Controller, several major tasks remain. First the JANUS/ADA language interface to the MCORTEX system must be completed, so that the Radar Scheduler model can be run and tested in a true concurrent multi-microprocessor environment. The information gained from this will be valuable to the future programmers of the remaining processes. Next the Track Processing portion of the Radar Control Group should be implemented. This portion is expected to be numerically intensive and should provide a good basis for testing the performance of the code produced by the JANUS/ADA compiler and the interaction of processes on seperate processors.

When the remainder of the processes have been completed, the Radar Controller model can be evaluated as a whole. At that time actual efficiency measurements can be made of the Radar Controller model and the effects of the Ada programming language in a real-time multiprocessor programming environment can be asessed.

APPENDIX A

COMMON MEMORY INTERFACE SOURCE CODE

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 3 Dec 86
-- MODULE TYPE: External data structure
-- PURPOSE: Common memory interface to: Radar Scheduler, Search Management,
-- and Track Management processes
-- NAME: Priority Event List ... TAB0.LIB

-- This data structure is the list of radar events in priority order.
-- Each element in the list holds information on the event queue it
-- points to.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH tab1,tab2;

PACKAGE tab0 IS

    lo_enhnc: CONSTANT INTEGER:= 4;
    max_pri: CONSTANT INTEGER:= 26;

    TYPE QueType IS (search,special_request,track,missile);

    USE tab1,tab2;

    TYPE BeamQue IS RECORD
        CASE kind: QueType IS
            WHEN search | special_request =>
                Snode: SearchPtr;              -- see TAB1.LIB
            WHEN track | missile =>
                Tnode: TrkPtr;                 -- see TAB2.LIB
            WHEN others =>
                null;
        END CASE;
    END RECORD;

    TYPE pri_lst_info IS RECORD
        status    :BOOLEAN;
        eventnm   :string(40);
        max_nodes:INTEGER;
        que_id    :QueType;
        que_ptr   :BeamQue;
        enhnc     :BOOLEAN;
        b_pri     :INTEGER;
        c_pri     :INTEGER;
        ltx       :INTEGER;
        allwd_ltx:INTEGER;
        slct_flg :BOOLEAN;
        nxt       :INTEGER;      -- pointer to next priority in list
    END RECORD;

    TYPE PriLstPtr IS ACCESS pri_lst_info;

    TYPE PriLstArray IS ARRAY(1..max_pri) OF PriLstPtr;

    pri_lst : PriLstArray;

END tab0;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 3 Dec 86
-- MODULE TYPE: External data structure
-- PURPOSE: Common memory interface to: Radar Scheduler, Search Management,
-- and Track Management processes
-- NAME: Priority Event List ... TAB0.PKG

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

PACKAGE BODY tab0 IS

    i:INTEGER;

BEGIN
    FOR i IN 1..max_pri LOOP
        pri_lst(i):= NEW pri_list_info;
    END LOOP;
END tab0;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 30 Sep 86
-- MODULE TYPE: external table
-- PURPOSE: Common memory interface
-- NAME: Search Node...TAB1.LIB

-- This interface is a search beam node. It acts as a template for the
-- nodes which make up the horizon search, above horizon search and
-- special request queues.
-- The Search Management Process fills the queue and the radar Scheduler
-- Process empties it.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

PACKAGE tab1 IS

    TYPE BeamPosition IS RECORD
        azim : INTEGER;
        elev : INTEGER;
    END RECORD;

    TYPE SrchData IS RECORD
        mode        : INTEGER;
        bid         : string(3);
        beam_posit  : BeamPosition;
        inst_rge    : INTEGER;
        stc_data    : INTEGER;
        doct_blnk_gte: INTEGER;
        cltr_blnk_gte: INTEGER;
        eof_indic   : BOOLEAN;
        req_id      : INTEGER;
    END RECORD;

    TYPE SrchDatPtr IS ACCESS SrchData;

    TYPE SearchNode;
    TYPE SearchPtr IS ACCESS SearchNode;
    TYPE SearchNode IS RECORD
        info: SrchDatPtr;
        nxt : SearchPtr;
    END RECORD;

    srch_node: SearchPtr;

END tab1;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 24 Jun 86
-- MODULE TYPE: external table
-- PURPOSE: common memory interface
-- NAME: Track Node ... TAB2.LIB

-- This data structure is a track request beam node. It acts as an overlay
-- for the track request queues resident in the Priority List. The Track
-- Management Process fills the queues with requests for track beams and
-- the Radar Scheduler Process empties them as the track beams are scheduled.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH tab7;

PACKAGE tab2 IS

    USE tab7;

    TYPE TrkInfo IS RECORD
        mode    : INTEGER;
        bid     : string(3);
        p_trk: PtrTrkFile;  -- See TAB7.LIB
    END RECORD;

    TYPE TrackNode;
    TYPE TrkPtr IS ACCESS TrackNode;
    TYPE TrackNode IS RECORD
        info   : TrkInfo;
        nxt    : TrkPtr;
    END RECORD;

    trk_node: TrkPtr;

END tab2;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 24 Jun 86
-- MODULE TYPE: external table
-- PURPOSE: common memory interface
-- NAME: I_to_R ... TAB3.LIB

-- This table is an interface data structure which communicates SPY-1 radar
-- status to the Radar Scheduling Process.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

PACKAGE tab3 IS

    TYPE SPY1_status IS RECORD
        resynch_time : INTEGER;
        loop_con     : INTEGER;
        xmsn_status  : BOOLEAN;
    END RECORD;

    TYPE StatPtr IS ACCESS SPY1_status;
    I_to_R: StatPtr;

END tab3;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 24 Jun 86
-- MODULE TYPE: external table
-- PURPOSE: common memory interface
-- NAME: A_to_R ...TAB4.LIB

-- This interface data structure communicates RCS commands to the Radar
-- Scheduler to aid in the formating of the radar dwell buffer.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

PACKAGE tab4 IS

    TYPE InhibitRegion IS RECORD
        start_bng :INTEGER;
        stop_bng  :INTEGER;
    END RECORD;

    TYPE OpDoct IS RECORD
        mtrks     :INTEGER;
        mintvls   :INTEGER;
        dply_rect :INTEGER;
    END RECORD;

    TYPE RCS_command IS RECORD
        pwr_flg        : BOOLEAN;
        rad_pwr_option: BOOLEAN;
        rad_inhibt_regions: array(1..5) OF InhibitRegion;
        op_doct        : OpDoct;
    END RECORD;

    TYPE Ptr_A_to_R IS ACCESS RCS_command;
    A_to_R : Ptr_A_to_R;

END tab4;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 22 Oct 86
-- MODULE TYPE: external table
-- PURPOSE: common memory interface
-- NAME: A_to_R ...TAB4.PKG

-- This interface data structure communicates RCS commands to the Radar
-- Scheduler to aid in the formating of the radar dwell buffer.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

PACKAGE BODY tab4 IS

BEGIN

    A_to_R:= NEW RCS_command;

END tab4;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 11 Oct 86
-- MODULE TYPE: External Table
-- PURPOSE: Common Memory Interface
-- NAME: C_to_R ... TAB5.LIB

-- This table is an interface between the C&D User Services Process
-- and the Radar Scheduling Process.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

PACKAGE tab5 IS

    rdr_silnce: BOOLEAN;

END tab5;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 11 Oct 86
-- MODULE TYPE: External Table
-- PURPOSE: Common Memory Interface
-- NAME: B_to_R ... TAB6.LIB

-- This interface data structure communicates information concerning
-- array face limits and ships motion matrix to the Radar Scheduler
-- Process. The information is developed in the Beam Stabilization
-- Process.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

PACKAGE tab6 IS

    TYPE L_R_Limits IS RECORD
        limit_left: INTEGER;
        right_limit: INTEGER;
    END RECORD;

    TYPE FaceLimits IS ARRAY(1..4) OF L_R_Limits;

    TYPE MotionMatrix IS RECORD
        x_ship_dot: INTEGER;
        y_ship_dot: INTEGER;
        roll      : INTEGER;
        pitch     : INTEGER;
        yaw       : INTEGER;
    END RECORD;

    TYPE B_R_Interface;
    TYPE BtoR IS ACCESS B_R_Interface;
    TYPE B_R_Interface IS RECORD
        face_antenna_limits: FaceLimits;
        ships_motion_matrix: MotionMatrix;
        azim_limit_slope   : INTEGER;
    END RECORD;

    B_to_R: BtoR;

END tab6;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 22 Oct 86
-- MODULE TYPE: external table
-- PURPOSE: common memory interface
-- NAME: Track File ...TAB7.LIB

-- This external table is the radar control system track file. The data
-- structure is a linked list based on a pointer which is passed between
-- processes which require access to the track file for data on tracks.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH Longops;

PACKAGE tab7 IS

    USE Longops;

    TYPE Position IS RECORD
        x          : INTEGER;
        y          : INTEGER;
        z          : INTEGER;
        slnt_rnge  : Long_Integer;   -- Long_Integer from Longops package.
        x_dot      : INTEGER;
        y_dot      : INTEGER;
        z_dot      : INTEGER;
        rge_dot    : INTEGER;
    END RECORD;

    TYPE TimAlwdDwl IS RECORD
        msw : INTEGER;
        lsw : INTEGER;
    END RECORD;

    TYPE UpdtPosit IS RECORD
        msw_rate_filter : INTEGER;
        lsw_rate_filter : INTEGER;
    END RECORD;

    TYPE PredTimLst IS RECORD
        msw_azim_elev : INTEGER;
        lsw_azim_elev : INTEGER;
    END RECORD;

    TYPE DetectRnge IS RECORD
        msw : INTEGER;
        lsw : INTEGER;
    END RECORD;

    TYPE TimDetect IS RECORD
        msw : INTEGER;
        lsw : INTEGER;
    END RECORD;

    TYPE TrkData IS RECORD
        mode             : INTEGER;
        bid              : string(3);
        priority         : INTEGER;
        posit            : Position;
        trk_xsitn_flag   : BOOLEAN;
        ctl_grp_trk_num  : INTEGER;
        ctsl             : INTEGER;
        xgte_bin_num     : INTEGER;
        pred_azim        : INTEGER;
        low_elev_trk_flg: BOOLEAN;
        log_ampld_est    : INTEGER;
```

```
        xmit_req_flg    : BOOLEAN;
        sim_tgt_flg     : BOOLEAN;
    END RECORD;

    TYPE TrkDatPtr IS ACCESS TrkData;

    TYPE TgtData IS RECORD
        tim_alwd_dwl    : TimAlwdDwl;
        mfar_trk_num    : INTEGER;
        wcs_idx         : INTEGER;
        updt_tim_posit  : UpdtPosit;
        pred_tim_lst    : PredTimLst;
        dwl_btwn_tim    : INTEGER;
        nxt_trk_xgte_bin: INTEGER;
        prev_trk_xgte_bin: INTEGER;
        grtst_noise_freq: INTEGER;
        least_noise_freq: INTEGER;
        valid_data_flg  : BOOLEAN;
        lost_trk_count  : INTEGER;
        nxt_tim_tbl_entry: INTEGER;

        --he=1,me=2,le=3,ale=4,mti=5,passv=6,missile=7,cvr_plse=8
        trk_mode        : INTEGER;

        rcvr_status_flg : BOOLEAN;
        typ1_passv_flg  : BOOLEAN;
        typ2_passv_flg  : BOOLEAN;
        drp_trk_flg     : BOOLEAN;

        --air=1,surface=2,clutter=3
        trk_class       : INTEGER;

        --hostile=1,friendly=2,unknown=3
        trk_id          : INTEGER;

        wcs_flg         : BOOLEAN;
        smth_ampld_count: INTEGER;
        rnge_accel_flg  : BOOLEAN;
        detect_rnge     : DetectRnge;
        tim_detect      : TimDetect;
        caus_lost_data_pt: INTEGER; --unknown pointer type?
    END RECORD;

    TYPE TrkFile;
    TYPE PtrTrkFile IS ACCESS TrkFile;
    TYPE TrkFile IS RECORD
        beam_data   : TrkDatPtr;
        tgt_data    : TgtData;
        nxt_trk     : PtrTrkFile;
    END RECORD;

    ptrk: PtrTrkFile;
    trk_file: PtrTrkFile;

END tab7;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 17 Nov 86
-- MODULE TYPE: external table
-- PURPOSE: common memory interface
-- NAME: Track File ...TAB7.PKG

-- This external table is the radar control system track file. The data
-- structure is a linked list based on a pointer which is passed between
-- processes which require access to the track file for data on tracks.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

PACKAGE BODY tab7 IS

BEGIN

    ptrk:= NEW TrkFile;
    ptrk.beam_data:= NEW TrkData;
    trk_file:= NEW TrkFile;
    trk_file.beam_data:= NEW TrkData;
    ptrk:= trk_file;
    ptrk.beam_data:= trk_file.beam_data;

END tab7;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 11 Oct 86
-- MODULE TYPE: External Table
-- PURPOSE: Common Memory Interface
-- NAME: F_to_R ... TAB8.LIB

-- This interface data structure contains the frequency and waveform
-- information required by the Radar Scheduler Process to complete
-- formatting of selected radar dwells.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

PACKAGE tab8 IS

    max_dwells: CONSTANT INTEGER:= 10; -- not previously defined, 10 ?

    TYPE mtiRec IS RECORD
        pri       : INTEGER;
        dwl_Length: INTEGER;
    END RECORD;

    TYPE msleRec IS RECORD
        uplnk_freq: INTEGER;
        dwn_freq  : INTEGER;
    END RECORD;

    TYPE WaveFormRec IS RECORD
        freq_chnl   : INTEGER;
        freq_band   : INTEGER;
        phase1_code: INTEGER;
        phase2_code: INTEGER;
        mti         : mtiRec;
        msle        : msleRec;
        inhib_freq_chnl: INTEGER;
    END RECORD;

    TYPE F_to_R IS ACCESS WaveFormRec;

    TYPE InfoWaveForm IS ARRAY(1..max_dwells) OF F_to_R;

    waveform: InfoWaveForm;

END tab8;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 11 Oct 86
-- MODULE TYPE: External Table
-- PURPOSE: Common Memory Interface
-- NAME: F_to_R ... TAB8.PKG

-- This interface data structure contains the frequency and waveform
-- information required by the Radar Scheduler Process to complete
-- formatting of selected radar dwells.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

PACKAGE BODY  tab8 IS

    i: INTEGER;

BEGIN
    FOR i IN 1..max_dwells LOOP
        waveform(i):= NEW WaveFormRec;
    END LOOP;
END tab8;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 11 Oct 86
-- MODULE TYPE: External Table
-- PURPOSE: Common Memory Interface
-- NAME: L_to_R ... TAB9.LIB

-- This interface data structure contains the flag which indicates
-- reset time for the track counter. This flag is used by the Radar
-- Scheduler Process to format track dwells.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

PACKAGE tab9 IS

    trk_tim_ctr_reset: BOOLEAN;

END tab9;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 11 Oct 86
-- MODULE TYPE: External Table
-- PURPOSE: Common Memory Interface
-- NAME: Missile Downlink Message ... TAB10.LIB

-- This table holds the communication information required for
-- downlink data from own ship's SM-2 missiles.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

PACKAGE tab10 IS

    num_mssl_msgs: CONSTANT INTEGER:= 10; -- not previously defined, 10?

    TYPE DwnlinkRec IS RECORD
        prt_one  :INTEGER;
        prt_two  :INTEGER;
        prt_three:INTEGER;
        prt_four :INTEGER;
        prt_five :INTEGER;
        prt_six  :INTEGER;
        prt_seven:INTEGER;
        prt_eight:INTEGER;
    END RECORD;

    TYPE PtrMsslDwnlnk IS ACCESS DwnlinkRec;

    TYPE DwnlnkArray IS ARRAY(1..num_mssl_msgs) OF PtrMsslDwnlnk;

    mssl_dwnlnk: DwnlnkArray;

END tab10;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 11 Oct 86
-- MODULE TYPE: External Table
-- PURPOSE: Common Memory Interface
-- NAME: Missile Downlink Message ... TAB10.PKG

-- This table holds the communication information required for
-- downlink data from own ship's SM-2 missiles.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

PACKAGE BODY tab10 IS

    i: INTEGER;

BEGIN
    FOR i IN 1.. num_mssl_msgs LOOP
        mssl_dwnlnk(i):= NEW DwnlinkRec;
    END LOOP;
END tab10;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 11 Oct 86
-- MODULE TYPE: External Table
-- PURPOSE: Common Memory Interface
-- NAME: R_to_S ... TAB11.LIB

-- This interface data structure contains the replenishment flags
-- which inform the Search Management Process which search queues
-- require filling. The Radar Scheduler Process sets the flags as
-- necessary when it has used a preset number of search request
-- beams.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);


PACKAGE tab11 IS

    TYPE RtoS IS RECORD
        hs_que_repln: BOOLEAN;
        ahs_que_repln: BOOLEAN;
    END RECORD;

    TYPE RtoSPtr IS ACCESS RtoS;

    R_to_S: RtoSPtr;

END tab11;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 11 Oct 86
-- MODULE TYPE: External Table
-- PURPOSE: Common Memory Interface
-- NAME: R_to_S ... TAB11.PKG

-- This interface data structure contains the replenishment flags
-- which inform the Search Management Process which search queues
-- require filling. The Radar Scheduler Process sets the flags as
-- necessary when it has used a preset number of search request
-- beams.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

PACKAGE BODY tab11 IS

BEGIN

    R_to_S:= NEW RtoS;

END tab11;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 30 Sep 86
-- MODULE TYPE: external table
-- PURPOSE: common memory interface
-- NAME: R_to_O ... TAB32.LIB

-- This table is the interface between the Radar Scheduler Process and the
-- Radar Output Process.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

PACKAGE tab32 IS

    TYPE DwlData IS RECORD
        mode                  : INTEGER;
        face                  : INTEGER;
        sub_mode              : ARRAY(1..2) OF INTEGER;
        dwl_idx               : INTEGER;
        beam_purpose          : INTEGER;
        dwl_start_idx         : INTEGER;
        doct_unblnk_gates     : INTEGER;
        clutter_unblnk_gates: INTEGER;
    END RECORD;

    TYPE RtoO;
    TYPE PtrRtoO IS ACCESS RtoO;
    TYPE RtoO IS RECORD
        dwl_data : DwlData;
        link     : PtrRtoO;
    END RECORD;

    R_to_O: PtrRtoO;

    TYPE ROPtrArray IS ARRAY(1..2) OF PtrRtoO;
    ptr_r_to_o: ROPtrArray;

END tab32;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 23 Oct 86
-- MODULE TYPE: external table
-- PURPOSE: common memory interface
-- NAME: R_to_O ... TAB32.PKG

-- This table is the interface between the Radar Scheduler Process and the
-- Radar Output Process.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

PACKAGE BODY tab32 IS

BEGIN

    ptr_r_to_o(1):= NEW RtoO;
    ptr_r_to_o(2):= NEW RtoO;
    ptr_r_to_o(1).link:= null;
    ptr_r_to_o(2).link:= null;

END tab32;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 30 Sep 86
-- MODULE TYPE: data buffer
-- PURPOSE: internal data structure
-- NAME: Output Control Channel Buffer ...TAB40.LIB


-- This interface data structure is a buffer between the Radar Control
-- program and the Radar Channel program.  It communicates with the
-- Radar Scheduler and Radar Output processes.  The Radar Channel program
-- uses the data to process commands for the SPY-1 radar.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

PACKAGE tab40 IS

    TYPE CntrlWord IS RECORD
        rdr_xmsn_on          : BOOLEAN;
        adj_detect_enable    : BOOLEAN;
        sdlb_canx_on         : BOOLEAN;
        chnl_a_video         : BOOLEAN;
        chnl_b_video         : BOOLEAN;
        freq_group_slct      : BOOLEAN;
        rng_scaling_enable   : BOOLEAN;
        t1_subchnl_disable   : BOOLEAN;
        t2_subchnl_disable   : BOOLEAN;
        t3_subchnl_disable   : BOOLEAN;
        t4_subchnl_disable   : BOOLEAN;
        subchnl_weight_enable: BOOLEAN;
        cltr_lock_ops        : BOOLEAN;
        cltr_lock_err        : BOOLEAN;
    END RECORD;

    TYPE FreqGroupSelect IS RECORD
        pri_mode_a : INTEGER;
        b_submode  : INTEGER;
        c_submode  : INTEGER;
    END RECORD;

    TYPE OaType IS RECORD
        cntrl_word        : CntrlWord;
        freq_group_select : FreqGroupSelect;
        data_block_code   : INTEGER;
    END RECORD;

    TYPE ObType IS RECORD
        detect1_thrsld : INTEGER;
        detect2_thrsld : INTEGER;
        detect3_thrsld : INTEGER;
        sdlb_thrsld    : INTEGER;
    END RECORD;

    TYPE OcType IS RECORD
        mnlb_thrsld1      : INTEGER;
        cvr_pulse_thrsld1 : INTEGER;
        sat1_thrsld       : INTEGER;
        sat2_thrsld       : INTEGER;
    END RECORD;

    TYPE OdType IS RECORD
        ratio_thrsld : INTEGER;
        limit_thrsld : INTEGER;
        cltr_thrsld  : INTEGER;
    END RECORD;

    TYPE OeType IS RECORD
```

```
    comptr_lock_init : INTEGER;
    trunc1_thrsld    : INTEGER;
    trunc2_thrsld    : INTEGER;
END RECORD;

TYPE OfType IS RECORD
    phse1_code : INTEGER;
    phse2_code : INTEGER;
    phse3_code : INTEGER;
    phse4_code : INTEGER;
END RECORD;

TYPE OgType IS RECORD
    fdbk1 : INTEGER;
    fdbk2 : INTEGER;
    fdbk3 : INTEGER;
    fdbk4 : INTEGER;
END RECORD;

TYPE OhType IS RECORD
    pri1_mti : INTEGER;
    pri2_mti : INTEGER;
END RECORD;

TYPE OiType IS RECORD
    cntrl_bit        : BOOLEAN;
    dwl_1_start_time : INTEGER;
    dwl_2_start_time : INTEGER;
END RECORD;

TYPE OjType IS RECORD
    cntrl_bit           : BOOLEAN;
    doct_1_unblnk_start : INTEGER;
    subchnl_freq_group  : INTEGER;
END RECORD;

TYPE OkType IS RECORD
    cntrl_bit           : BOOLEAN;
    doct_1_unblnk_stop  : INTEGER;
    fixed_atten_cntrl   : INTEGER;
    stc_cntrl           : INTEGER;
END RECORD;

TYPE OlType IS RECORD
    cntrl_bit           : BOOLEAN;
    doct_2_unblnk_start : INTEGER;
    xmit_freq           : INTEGER;
    rcv_freq            : INTEGER;
END RECORD;

TYPE OmType IS RECORD
    cntrl_bit           : BOOLEAN;
    doct_2_unblnk_stop  : INTEGER;
    face                : INTEGER;
    fore_beam_alarm_inhib: INTEGER;
    cos_alpha1_mn_array : INTEGER;
END RECORD;

TYPE OnType IS RECORD
    cntrl_bit           : BOOLEAN;
    cltr_1_unblnk_start : INTEGER;
    aft_beam_alert_inhib: INTEGER;
    cos_alpha2_sdlb_blnk_ary: INTEGER;
END RECORD;

TYPE OoType IS RECORD
    cntrl_bit           : BOOLEAN;
    cltr_1_unblnk_start : INTEGER;
    cos_beta1_mn_array  : INTEGER;
```

```
    END RECORD;

    TYPE OpType IS RECORD
        cntrl_bit           : BOOLEAN;
        cltr_2_unblk_strt   : INTEGER;
        cos_beta2_sdlb_blnk_ary: INTEGER;
    END RECORD;

    TYPE OqType IS RECORD
        cntrl_bit           : BOOLEAN;
        cltr_2_unblk_stop   : INTEGER;
        elev_sector         : INTEGER;
        dply_ctrl           : INTEGER;
        trk_num             : INTEGER;
    END RECORD;

    TYPE OrType IS RECORD
        trk_gate_strt : INTEGER;
        dply_elev     : INTEGER;
    END RECORD;

    TYPE OsType IS RECORD
        video_extnt   : INTEGER;
        dply_grp_slct : INTEGER;
        dply_azim     : INTEGER;
    END RECORD;

    TYPE OtType IS RECORD
        otmsb : INTEGER;
        otlsb : INTEGER;
    END RECORD;
    TYPE OtArray IS ARRAY(1..16) OF OtType;

    TYPE OccbType;
    TYPE PtrOccb IS ACCESS OccbType;
    TYPE OccbType IS RECORD
        oa    : OaType;
        ob    : ObType;
        oc    : OcType;
        od    : OdType;
        oe    : OeType;
        o_f   : OfType;
        og    : OgType;
        oh    : OhType;
        oi    : OiType;
        oj    : OjType;
        ok    : OkType;
        ol    : OlType;
        om    : OmType;
        on    : OnType;
        oo    : OoType;
        op    : OpType;
        oq    : OqType;
        o_r   : OrType;
        os    : OsType;
        ot    : OtArray;
        link : PtrOccb;
    END RECORD;

    occb: PtrOccb;

    TYPE OCCBPtrArray IS ARRAY(1..2) OF PtrOccb;
    occb_ptr: OCCBPtrArray;

END tab40;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 23 Oct 86
-- MODULE TYPE: data buffer
-- PURPOSE: internal data structure
-- NAME: Output Control Channel Buffer ...TAB40.PKG


-- This interface data structure is a buffer between the Radar Control
-- program and the Radar Channel program.  It communicates with the
-- Radar Scheduler and Radar Output processes.  The Radar Channel program
-- uses the data to process commands for the SPY-1 radar.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

PACKAGE BODY tab40 IS

BEGIN

    occb_ptr(1):= NEW OccbType;
    occb_ptr(2):= NEW OccbType;
    occb_ptr(1).link:= null;
    occb_ptr(2).link:= null;

END tab40;
```

APPENDIX B

GLOBAL SOURCE CODE

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 22 Oct 86
-- MODULE TYPE: Global data
-- PURPOSE: System's Data Structure declarations
-- NAME: GLOBAL DECLARATIONS...GLOBAL.LIB

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

PACKAGE global IS

    --SES CONSTANTs

    numb_events: CONSTANT INTEGER := 28;
    buff_size: CONSTANT INTEGER :=10;
    infinity: CONSTANT INTEGER := 32000;
    numb_procs: CONSTANT INTEGER := 21;

    --process identifiers

    a_pid: CONSTANT INTEGER := 1;
    c_pid: CONSTANT INTEGER := 2;
    b_pid: CONSTANT INTEGER := 3;
    d_pid: CONSTANT INTEGER := 4;
    e_pid: CONSTANT INTEGER := 5;
    f_pid: CONSTANT INTEGER := 6;
    k_pid: CONSTANT INTEGER := 7;
    l_pid: CONSTANT INTEGER := 8;
    m_pid: CONSTANT INTEGER := 9;
    w_pid: CONSTANT INTEGER := 10;
    x_pid: CONSTANT INTEGER := 11;
    h_pid: CONSTANT INTEGER := 12;
    o_pid: CONSTANT INTEGER := 13;
    i_pid: CONSTANT INTEGER := 14;
    p_pid: CONSTANT INTEGER := 15;
    t_pid: CONSTANT INTEGER := 16;
    s_pid: CONSTANT INTEGER := 17;
    r_pid: CONSTANT INTEGER := 18;
    main_id: CONSTANT INTEGER := 19;
    idle_id: CONSTANT INTEGER := 20;

    --event count identifiers, reserved event count identifiers: 0,1,2,6,20

    main_event_id: CONSTANT INTEGER := 1;
    display_evc_id: CONSTANT INTEGER := 2;
    time_count_id: CONSTANT INTEGER := 6;
    idle_event_id: CONSTANT INTEGER := 20;
    ea_id: CONSTANT INTEGER := 3;
    ec_id: CONSTANT INTEGER := 4;
    eb_id: CONSTANT INTEGER := 5;
    ed_id: CONSTANT INTEGER := 7;
    ee_id: CONSTANT INTEGER := 8;
    ef_id: CONSTANT INTEGER := 9;
    ek_id: CONSTANT INTEGER := 10;
    el_id: CONSTANT INTEGER := 11;
    em_id: CONSTANT INTEGER := 12;
    ew_id: CONSTANT INTEGER := 13;
    ex_id: CONSTANT INTEGER := 14;
    eh_id: CONSTANT INTEGER := 15;
```

```
    eo_id: CONSTANT INTEGER := 16;
    ei_id: CONSTANT INTEGER := 17;
    ep_id: CONSTANT INTEGER := 18;
    et_id: CONSTANT INTEGER := 19;
    es_id: CONSTANT INTEGER := 21;
    er_id: CONSTANT INTEGER := 22;

    --common service routines

    FUNCTION clock RETURN INTEGER;
    FUNCTION rand RETURN INTEGER;
    PROCEDURE ipl;
    PROCEDURE await(proc_id,event_id,event_value: IN INTEGER);
    PROCEDURE advance(proc_id,event_id: IN INTEGER);
    FUNCTION ticket RETURN INTEGER;
    PROCEDURE disable;
    PROCEDURE enable;

  --System's Data Objects

  TYPE EveValArray IS ARRAY (0..numb_events) OF INTEGER;
  event_val_cnt :EveValArray;

  TYPE process IS RECORD
      status    : INTEGER;
      context   : INTEGER;
      count     : INTEGER;
      event     : INTEGER;
  END RECORD;
  TYPE ProTabArray IS ARRAY (0..numb_procs) OF process;

  pros_tble : ProTabArray;

END global;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 26 Nov 86
-- MODULE TYPE: global package body
-- PURPOSE: to define common service routines
-- NAME: global ... GLOBAL.PKG

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH Longops,tab0,tab1,tab2,tab7;

PACKAGE BODY global IS

    USE Longops,tab0,tab1,tab2,tab7;

-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 1 Dec 86
-- MODULE TYPE: common service routine
-- PURPOSE: generate a random number
-- NAME: RAND ...csr5

-- This pseudo-random number generator uses the congruence algorithm
-- to generate a list of random numbers with a period approximately
-- equal to "t" . The equation is of the form:
--          X(n+1) = mod ((A * X(n) + B) T)

    x: INTEGER:= -1;

    FUNCTION rand RETURN INTEGER IS

        a: INTEGER := 667;
        b: INTEGER := 37;
        t: INTEGER := 31099;
        seed: INTEGER := 19879;
        y: Long_Integer;

    BEGIN
        IF x=-1 THEN
            x:= seed;
        END IF;

        -- compute the random number
        y:= Ladd(Lmul(Lint(a),Lint(x)),Lint(b));
        x:=L_to_int(Lmod(y,Lint(t)));
        RETURN x;
    END rand;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 30 Nov 86
-- MODULE TYPE: external initialization routine
-- PURPOSE: initialize the priority event list for the Radar Scheduler Model
-- NAME: Initialize the Radar Event Priority List...csr6
--
-- This routine initializes the Priority Event List which is used by the
-- Radar Scheduler, Search Management and Track Management processes to
-- request and schedule radar dwells.


    PROCEDURE ipl IS

        i: INTEGER;

    BEGIN
        for i in 1..max_pri LOOP
            pri_lst(i).status:= false;
            pri_lst(i).b_pri:= 1;
            pri_lst(i).c_pri:= 1;
            pri_lst(i).ltx:= 0;
            pri_lst(i).slct_flg:= false;
            pri_lst(i).nxt:= i+1;

            IF ((i<=8) OR ((i>=10) AND (i<=13))) THEN
                pri_lst(i).allwd_ltx:= 100;
            ELSIF ((i>=14) AND (i<=21)) THEN
                pri_lst(i).allwd_ltx:= 500;
            ELSIF (i=9) THEN
                pri_lst(i).allwd_ltx:= 25;
            ELSIF (i=22) THEN
                pri_lst(i).allwd_ltx:= 50;
            ELSE
                pri_lst(i).allwd_ltx:= infinity;
            END IF;

            IF ((i<lo_enhnc) OR (i>22)) THEN
                pri_lst(i).enhnc:= false;
            ELSE
                pri_lst(i).enhnc:= true;
            END IF;

            IF ((i<=3) OR ((i>=10) AND (i<=13)) OR (i>=23)) THEN
                pri_lst(i).max_nodes:= 5;
                pri_lst(i).que_id:= special_request;
                pri_lst(i).que_ptr.kind:= special_request;
                pri_lst(i).que_ptr.Snode:= NEW SearchNode;
                pri_lst(i).que_ptr.Snode.nxt:= null;
            ELSIF ((i=4) OR ((i>=6) AND (i<=8)) OR ((i>=14) AND (i<=21))) THEN
                pri_lst(i).max_nodes:= 5;
                pri_lst(i).que_id:= track;
                pri_lst(i).que_ptr.kind:= track;
                pri_lst(i).que_ptr.Tnode:= NEW TrackNode;
                pri_lst(i).que_ptr.Tnode.info.p_trk:= NEW TrkFile;
                pri_lst(i).que_ptr.Tnode.info.p_trk.beam_data:= NEW TrkData;
                pri_lst(i).que_ptr.Tnode.nxt:= null;
            ELSIF ((i=9) OR (i=22)) THEN
                pri_lst(i).max_nodes:= 10;
                pri_lst(i).que_id:= search;
                pri_lst(i).que_ptr.kind:= search;
                pri_lst(i).que_ptr.Snode:= NEW SearchNode;
                pri_lst(i).que_ptr.Snode.info:= NEW SrchData;
                pri_lst(i).que_ptr.Snode.nxt:= null;
            ELSE
                pri_lst(i).max_nodes:= 2;
                pri_lst(i).que_id:= missile;
                pri_lst(i).que_ptr.kind:= missile;
                pri_lst(i).que_ptr.Tnode:= NEW TrackNode;
                pri_lst(i).que_ptr.Tnode.nxt:= null;
```

```
        END IF;

    END LOOP;

    --reset "nxt" on last element in Priority List to point to 0
    pri_lst(max_pri).nxt:= 0;

    -- assign event names to each priority
    pri_lst(1).eventnm:= "A-EVENT - ECM BURNTHROUGH";
    pri_lst(2).eventnm:= "B-EVENT - TARGET DEFINITION";
    pri_lst(3).eventnm:= "C-EVENT - SPECIAL TEST";
    pri_lst(4).eventnm:= "D-EVENT - ENGAGED HOSTILE TARGET";
    pri_lst(5).eventnm:= "E-EVENT - OWN SM-2 MISSILE GUIDANCE";
    pri_lst(6).eventnm:= "F-EVENT - PRE-ENGAGED HOSTILE TARGET";
    pri_lst(7).eventnm:= "G-EVENT - HIGH PRI TRACK TRANSITION";
    pri_lst(8).eventnm:= "H-EVENT - HIGH PRI TRACK CONFIRMATION";
    pri_lst(9).eventnm:= "I-EVENT - HORIZON SEARCH";
    pri_lst(10).eventnm:= "J-EVENT - SPECIAL ECM BURNTHROUGH";
    pri_lst(11).eventnm:= "K-EVENT - SPECIAL TARGET DEFINITION";
    pri_lst(12).eventnm:= "L-EVENT - SPECIAL MANUAL SCAN";
    pri_lst(13).eventnm:= "M-EVENT - SPECIAL TARGET ACQUISITION";
    pri_lst(14).eventnm:= "N-EVENT - CONFIRMED HOSTILE TRACK";
    pri_lst(15).eventnm:= "O-EVENT - ASSUMED HOSTILE TRACK";
    pri_lst(16).eventnm:= "P-EVENT - UNEVALUATED TRACK";
    pri_lst(17).eventnm:= "Q-EVENT - CONTROLLED FRIENDLY TRACK";
    pri_lst(18).eventnm:= "R-EVENT - TRACK CONFIRMATION";
    pri_lst(19).eventnm:= "S-EVENT - TRACK TRANSITION";
    pri_lst(20).eventnm:= "T-EVENT - ASSUMED FRIENDLY TRACK";
    pri_lst(21).eventnm:= "U-EVENT - CONFIRMED FRIENDLY TRACK";
    pri_lst(22).eventnm:= "V-EVENT - ABOVE HORIZON SEARCH";
    pri_lst(23).eventnm:= "W-EVENT - SPECIAL ABOVE HORIZON SEARCH";
    pri_lst(24).eventnm:= "X-EVENT - SIMULATION DWELL";
    pri_lst(25).eventnm:= "Y-EVENT - DIAGNOSTIC DWELL";
    pri_lst(26).eventnm:= "Z-EVENT - DUMMY DWELL";

END ipl;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 24 Jun 86
-- MODULE TYPE: common service routine
-- PURPOSE: simulates a real time millisecond clock
-- NAME: clock ...csr7

-- This common service routine simulates a real time millisecond clock.
-- when the routine is invoked the variable "time" is incremented by
-- one. The new value of time is then RETURNed to the invoking PROCEDURE.


    time: INTEGER;

    FUNCTION clock RETURN INTEGER IS

    BEGIN
        time:=time + 1;
        RETURN time;
    END clock;
```

```
-- The following functions and procedures are only stubs for testing

    PROCEDURE await(proc_id,event_id,event_val:IN INTEGER) IS
    BEGIN
      null;
    END await;

    PROCEDURE advance(proc_id,event_id:IN INTEGER) IS
    BEGIN
       null;
    END advance;

    FUNCTION ticket RETURN INTEGER IS
    BEGIN
       RETURN 0;
    END ticket;

    PROCEDURE disable IS
    BEGIN
       null;
    END disable;

    PROCEDURE enable IS
    BEGIN
       null;
    END enable;

-- initialization
BEGIN

    time:= -1;

END global;
```

APPENDIX C

RADAR SCHEDULER SOURCE CODE

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 29 Aug 86
-- MODULE TYPE: Process  vers 6.0
-- PURPOSE: Model the Radar Scheduler Function
-- NAME: Radar Scheduler ... RRCM.LIB

-- The Radar Scheduler selects requested beams from queues generated
-- by the Search Management and Track Management functions.  The
-- selected beams are then processed and formatted into dwells.  The
-- dwells are transmitted to the Radar Output function and are packed
-- into the Channel Output Buffer.  Beams are selected for dwell
-- processing based on a priority scheme.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

PACKAGE rrcm IS

    PROCEDURE radar_scheduler;

END rrcm;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 2 Dec 86
-- MODULE TYPE: Process  vers 6.0
-- PURPOSE: Model the Radar Scheduler Function
-- NAME: Radar Scheduler ... RRCM.PKG

-- The Radar Scheduler selects requested beams from queues generated
-- by the Search Management and Track Management functions.  The
-- selected beams are then processed and formatted into dwells.  The
-- dwells are transmitted to the Radar Output function and are packed
-- into the Channel Output Buffer.  Beams are selected for dwell
-- processing based on a priority scheme.


    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH Io,Util,global,tab0,tab1,tab2,tab4,tab7,tab32,tab40,rsm0,rsm2,
     rsm3,rsm5,rsm9,rsm10,rsm12,rsm13,rsm14;

PACKAGE BODY rrcm IS

    USE Io,Util,global,tab0,tab1,tab2,tab4,tab7,tab32,tab40,rsm0,rsm2,
        rsm3,rsm5,rsm9,rsm10,rsm12,rsm13,rsm14;

    PROCEDURE radar_scheduler IS

        i: INTEGER:= 0;
        j: INTEGER:= 2;

        --beam selection module CONSTANTs

        rdrint: CONSTANT INTEGER:= 21;
        srch_que: CONSTANT INTEGER:= 1;
        sr_que: CONSTANT INTEGER:= 2;
        srch_dwls: CONSTANT INTEGER:= 9;
        sr_dwls: CONSTANT INTEGER:= 2;
        rdr_rsrcs: CONSTANT INTEGER:= 100;
        trk_que: CONSTANT INTEGER:= 3;
        mssl_que: CONSTANT INTEGER:= 4;
        trk_dwls: CONSTANT INTEGER:= 10;
        mssl_dwls: CONSTANT INTEGER:= 5;

        rru       : INTEGER;
        sru       : INTEGER;
        sra       : INTEGER;
        tot_dwl_schd: INTEGER;
        tot_dwls  : INTEGER;
        srch_sra  : INTEGER;
        trk_sra   : INTEGER;
        mssl_sra  : INTEGER;
        sr_sra    : INTEGER;
        hwc       : boolean;

        et       : INTEGER;
        rtim     : INTEGER;
        oltim    : INTEGER;
        dltatim  : INTEGER;


    BEGIN

        Delete("RSOUT.Txt");
        Create(Text,"RSOUT.Txt",Write_Only);
        Open(Text,"RSOUT.Txt",Write_Only);

        WHILE (i < A_to_R.op_doct.mintvls) LOOP
```

```
i:= i + 1;
intvl_num:= i;

-- advance the radar loop event counts

advance(r_pid,es_id);
advance(r_pid,et_id);
rtim:= clock();    -- from csr7 in global.pkg

-- swap external dwell buffers
swap(buff_ptr,cm_ptr,j);  -- from rsm2.pkg

-- enhance event priorities
enhance(et);          -- from rsm3.pkg

-- resynchronization module never written
-- rsm4.pkg

-- begin beam selection
-- initialize the priority list traversal constraints
et:= 0;
rru:= 0;
tot_dwl_schd:= 0;
srch_sra:= srch_dwls;
trk_sra:= trk_dwls;
sr_sra:= sr_dwls;
mssl_sra:= mssl_dwls;
tot_dwls:= srch_dwls + trk_dwls + sr_dwls + mssl_dwls;

-- traverse the Priority List
ppl:= 1;
WHILE ((et < rdrint ) AND (rru < rdr_rsrcs) AND
          (tot_dwl_schd < tot_dwls) AND (ppl /= 0)) LOOP

    -- check for an empty queue
    IF pri_lst(ppl).status THEN

        -- traverse the priority event queue
        -- initialize the event queue traversal constraints
        sru:= 0;
        CASE pri_Lst(ppl).que_id IS
            WHEN search => sra:= srch_sra;
            WHEN special_request => sra:= sr_sra;
            WHEN track => sra:= trk_sra;
            WHEN missile => sra:= mssl_sra;
        END CASE;

        -- select a search type beam or a track type beam
        CASE pri_lst(ppl).que_id IS
            WHEN search | special_request =>

                --traverse the search event queue
                sp:= pri_lst(ppl).que_ptr.Snode;
                WHILE (sp /= null) LOOP

                    -- get a Radar Event node
                    r:= get_rect_node();   -- from RSM5B

                    -- insert beam information into the node
                    r.srch_dwl.ALL:= sp.info.ALL;
                    r.beamid:= r.srch_dwl.bid;

                     -- format the selected beams
                     -- do supplementary processing
                     -- rsm6 not writen
                     -- do face assignment
                     -- rsm7 not written
                     -- radar doctrine
                     -- rsm8 not writen
```

```
        -- satisfy the hardware constraints
        hw_constraint(pri_lst(ppl).que_id,rru,r.dru,hwc);

    IF hwc THEN

        -- indicate to the Priority List that this
        -- event has been scheduled
        pri_lst(ppl).slct_flg:= TRUE;

        -- fill the external table module
        fill_ext_tab(buff_ptr,occb_ptr(j),cm_ptr,
                     ptr_r_to_o(j),r.ocb_data); -- rsm10

        -- insert the dwell at the end of the Radar
        -- Event Control Table list
        llend(r,rect_ptr);  -- from rsm5a.pkg

        -- load the evaluation module
        -- rsm11 never written

        -- update the used resources
        IF (pri_lst(ppl).que_id = search) THEN
            srch_sra:= srch_sra - 1;
        ELSE
            sr_sra:= sr_sra - 1;
        END IF;
        tot_dwl_schd:= tot_dwl_schd + 1;
        sru:= sru + 1;

    ELSE
        -- hardware constraints have not been satisfied
        free_rect_node(r);  -- from rsm5c.pkg
    END IF;

    -- get next event in search or special request
    --queue
    sp:= sp.nxt;

END LOOP;

--traverse the search event queue
tp:= pri_lst(ppl).que_ptr.Tnode;
WHILE (sp /= null) LOOP

    -- get a Radar Event node
    r:= get_rect_node();  -- from rsm5b.pkg

    -- insert beam information into node
    r.trk_dwl.ALL:= tp.info.p_trk.beam_data.ALL;
    r.beamid:= tp.info.bid;

    -- format the selected beam
    -- do supplementary processing
    -- rsm6 not written
    -- do face assignment
    -- rsm7 not written
    -- radar doctrine module
    -- rsm8 not written

    -- satisfy the hardware constraints
    hw_constraint(pri_lst(ppl).que_id,rru,r.dru,hwc);

    IF hwc THEN

        -- indicate to the Priority List that this
        -- event has been scheduled
        pri_lst(ppl).slct_flg:= TRUE;
```

```
                                -- fill the external table module
                                fill_ext_tab(buff_ptr,occb_ptr(j),cm_ptr,
                                             ptr_r_to_o(j),r.ocb_data); -- rs

                                -- insert the dwell at the end of the Radar
                                -- Event Control Table list
                                llend(r,rect_ptr);  -- from rsm5a.pkg

                                -- load the evaluation module
                                -- rsm11 never written

                                -- update the used resources
                                IF (pri_lst(ppl).que_id = track) THEN
                                    trk_sra:= trk_sra - 1;
                                ELSE
                                    mssl_sra:= mssl_sra - 1;
                                END IF;
                                tot_dwl_schd:= tot_dwl_schd + 1;
                                sru:= sru + 1;

                            ELSE
                                -- hardware constraints have not been satisfied
                                free_rect_node(r);  -- from rsm5c.pkg
                            END IF;

                            -- get next event in track or missile queue
                            tp:= tp.nxt;

                        END LOOP;

                    END CASE;

                END IF;   -- end traverse event que & check for empty que

                -- compute time elapsed
                elapsed_time(et,rtim);    -- from rsm12.pkg

                -- point to next priority in the list
                ppl:= pri_lst(ppl).nxt;

            END LOOP;      -- end traverse priority list

            IF ((intvl_num MOD A_to_R.op_doct.dply_rect) = 0) THEN
                dump;  -- from rsm13
                Put("Completed interval : ");Put(intvl_num);New_Line;
            END IF;

            -- free memory for next interval
            free_memory(pool_ptr,rect_ptr);   -- from rsm14.pkg

        END LOOP;      -- end do one interval
        Close(Text);

    END radar_scheduler;

END rrcm;
```

```
-- OWNER:AEGIS Modeling Group
-- DATE OF LAST UPDATE: 23 Oct 86
-- MODULE TYPE: local data structure declarations  vers 4.0
-- PURPOSE: declare Radar Scheduler local data structures
-- NAME: Radar Scheduler Local Declarations ...RSM0.LIB

-- This file contains all data structures internal to the Radar Scheduler
-- Process.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH tab1,tab2,tab7,tab32,tab40;

PACKAGE rsm0 IS

    USE tab1,tab2,tab7,tab32,tab40;

    --beam selection module CONSTANTs

    rdrint: CONSTANT INTEGER:= 21;
    srch_que: CONSTANT INTEGER:= 1;
    sr_que: CONSTANT INTEGER:= 2;
    srch_dwls: CONSTANT INTEGER:= 9;
    sr_dwls: CONSTANT INTEGER:= 2;
    rdr_rsrcs: CONSTANT INTEGER:= 100;
    trk_que: CONSTANT INTEGER:= 3;
    mssl_que: CONSTANT INTEGER:= 4;
    trk_dwls: CONSTANT INTEGER:= 10;
    mssl_dwls: CONSTANT INTEGER:= 6;

    ppl:INTEGER;
    intvl_num:INTEGER;


    --Radar Event Control table node

    type OcbData IS RECORD
        ocb_purp        : INTEGER;
        xmit_flg        : boolean;
        face_assign     : INTEGER;
        pri_mti         : ARRAY(1..2) OF INTEGER;
        doct_blnk_gte   : INTEGER;
        cltr_blnk_gte   : INTEGER;
        mis_up_com      : INTEGER;
        mis_indx        : INTEGER;
        xmit_freq_chnl  : INTEGER;
        rcv_freq_chnl   : INTEGER;
        subchnl_freq_grp: INTEGER;
        phse_code       : INTEGER;
        fdbk_phsecode   : INTEGER;
        mjr_a_mode      : INTEGER;
        b_submode       : INTEGER;
        c_submode       : INTEGER;
        freq_grp_slct   : boolean;
        dwl_strt_ctl    : INTEGER;
        detect_thrslds  : ARRAY(1..3) OF INTEGER;
        trunc_thrslds   : ARRAY(1..2) OF INTEGER;
        dwl_idx_num     : INTEGER;
        elev_sctr       : INTEGER;
        dply_azim       : INTEGER;
        dply_elev       : INTEGER;
        vid_extnt       : INTEGER;
        rge_trk_gte_strt: INTEGER;
    END RECORD;
```

```
    type RadEveConTab;
    type RECTPtr is access RadEveConTab;
    type RadEveConTab IS RECORD
       srch_dwl      : SrchDatPtr;  -- See TAB1.LIB
       trk_dwl       : TrkDatPtr;   -- See TAB7.LIB
       ocb_data      : OcbData;
       beamid        : string(3);
       dru           : INTEGER;
       nxt_event     : RECTPtr;
    END RECORD;
    --end Radar Control Table node declaration

    sp: SearchPtr;     -- See TAB1.LIB
    tp: TrkPtr;        -- See TAB2.LIB
    r: RECTPtr;
    rect_ptr: RECTPtr;
    pool_ptr: RECTPtr;
    buff_ptr: PtrOccb;      -- See TAB40.LIB
    cm_ptr: PtrRto0;        -- See TAB32.LIB

END rsm0;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 30 Nov 86
-- MODULE TYPE: Radar Schedular Module  vers 4.0
-- PURPOSE: Initialize lists and variable for the Radar Schedular Process
-- NAME: Radar Schedular Initialization ...RSM0.PKG

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH tab1,tab2,tab7,tab32,tab40;

PACKAGE BODY rsm0 IS

    USE tab1,tab2,tab7,tab32,tab40;

    -- initialization module constants

    pool_length: CONSTANT INTEGER:= 22;
    buff_pool_length: CONSTANT:= 2;

    -- Procedure make_pool is derived from PL1 RSM1A module (MAKE POOL).
    -- This procedure creates a pool of available Radar Event Control
    -- Table nodes (see RSM0).

    PROCEDURE make_pool(FirstNode:IN OUT RECTPtr;numb_nodes:IN INTEGER) IS

        count: INTEGER;
        p,q: RECTPtr := NEW RadEveConTab;

    BEGIN
        p.nxt_event:= null;
        p.srch_dwl:= New SrchData;
        p.trk_dwl:= New TrkData;
        p:= FirstNode;
        p.ALL:= FirstNode.ALL;


        FOR count IN 2..numb_nodes LOOP
            q:= NEW RadEveConTab;
            q.srch_dwl:= New SrchData;
            q.trk_dwl:= New TrkData;
            q.nxt_event:= null;
            p.nxt_event:= q;
            p.nxt_event.ALL:= q.ALL;
            p:= p.nxt_event;
        END LOOP;

    END make_pool;
```

```
-- This procedure is derived from PL1 RSM1B (CREATE DWELL BUFFER POOLS).
-- This procedure creates two circular lists for each of the common
-- memory interfaced data structures (see TABLE32 & TABLE40) which
-- receive the formatted dwells from the Radar Scheduler Process.


PROCEDURE ex_buff_create(buff1:IN OUT OCCBPtrArray;buff2:IN OUT ROPtrArray;
                                 length: IN INTEGER) IS


    p1,q1:PtrOccb := NEW OccbType;
    p2,q2:PtrRtoO := NEW RtoO;
    ctr,j: INTEGER;

BEGIN
    FOR j IN 1..2 LOOP
        p1:= buff1(j);
        p1.link:= null;

        p2:= buff2(j);
        p2.link:= null;

        FOR ctr IN 1..length LOOP
            q1:= NEW OccbType;
            q1.link:= null;
            p1.link:= q1;

            q2:= NEW RtoO;
            q2.link:= null;
            p2.link:= q2;
        END LOOP;

        -- create circular lists

        q1.link:= buff1(j);
        q2.link:= buff2(j);
    END LOOP;

END ex_buff_create;
```

```
BEGIN
    pool_ptr:= NEW RadEveConTab;
    pool_ptr.nxt_event:= null;
    pool_ptr.srch_dwl:= NEW SrchData;
    pool_ptr.trk_dwl:= NEW TrkData;
    make_pool(pool_ptr, pool_length);
    buff_ptr:= NEW OccbType;
    buff_ptr.link:= null;
    cm_ptr:= NEW RtoO;
    cm_ptr.link:= null;
    ex_buff_create(occb_ptr,ptr_r_to_o,buff_pool_length);

    sp:= NEW SearchNode;
    sp.info:= NEW SrchData;
    sp.nxt:= null;
    tp:= NEW TrackNode;
    tp.info.p_trk:= NEW TrkFile;
    tp.info.p_trk.beam_data:= NEW TrkData;
    tp.nxt:= null;
    r:= NEW RadEveConTab;
    r.srch_dwl:= NEW SrchData;
    r.trk_dwl:=NEW TrkData;
    r.nxt_event:= null;
    rect_ptr:= NEW RadEveConTab;
    rect_ptr.srch_dwl:= NEW SrchData;
    rect_ptr.trk_dwl:= NEW TrkData;
    rect_ptr.nxt_event:= null;

END rsmO;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 30 Sep 86
-- MODULE TYPE: internal routine for the Radar Scheduler  vers 2.0
-- PURPOSE: swap output buffers for the next scheduling intervial
-- NAME: SWAP... RSM2.LIB

-- This module maintains the pointers to the two non-current dwell buffers
-- which are the destination of the formated dwells scheduled during the
-- current scheduling intervial.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH tab32,tab40;

PACKAGE rsm2 IS

    USE tab32,tab40;
    PROCEDURE swap(buff:IN OUT PtrOccb;cm:IN OUT PtrRtoO;index:IN OUT INTEGER);

END rsm2;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 30 Sep 86
-- MODULE TYPE: internal routine for the Radar Scheduler  vers 2.0
-- PURPOSE: swap output buffers for the next scheduling intervial
-- NAME: SWAP... RSM2.PKG

-- This module maintains the pointers to the two non-current dwell buffers
-- which are the destination of the formated dwells scheduled during the
-- current scheduling intervial.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH tab32,tab40;

PACKAGE BODY rsm2 IS

    USE tab32,tab40;

    PROCEDURE swap(buff:IN OUT PtrOccb;cm:IN OUT PtrRtoO;index:IN OUT INTEGER)
                IS


    BEGIN
        IF (index=2) THEN
            index:= 1;
        ELSE
            index:= 2;
        END IF;

        buff:= occb_ptr(index);
        cm:= ptr_r_to_o(index);

    END swap;

END rsm2;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 30 Sep 86
-- MODULE TYPE: Radar Scheduler Module  vers 3.0
-- PURPOSE: enhance and dehance priorities of events in the Radar Event
-- Priority List
-- NAME: enhance... RSM3.LIB

-- This module enhances the priorities of the radar events as listed in
-- the Priority List if certian conditions are met. The conditions are:
--    1.  The enhance flag must be set to true
--    2.  The ltx value must be greater than the allwd_ltx value

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

PACKAGE rsm3 IS

    PROCEDURE enhance(elapsed_time: IN INTEGER);

END rsm3;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 30 Sep 86
-- MODULE TYPE: Radar Scheduler Module  vers 3.0
-- PURPOSE: enhance and dehance priorities of events in the Radar Event
-- Priority List
-- NAME: enhance... RSM3.PKG

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH tab0;

PACKAGE BODY rsm3 IS

    USE tab0;


-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 26 Nov 86
-- MODULE TYPE: Radar Scheduler Subroutine
-- PURPOSE: remove, insert and reset the current priorities of Priority
-- List Events
-- NAME: Remove and Insert ... RSM3A

-- This module locates a request event node in the priority list,
-- removes it from its current priority, then locates its new position
-- in the priority list, and inserts it there.

    PROCEDURE ripl(curnt, new_p: IN INTEGER) IS

        p: INTEGER:= 1;
        b4: INTEGER:= 1;
        temp1, temp2: INTEGER;

    BEGIN
        IF (curnt /= new_p) THEN
        -- remove the node from its current position
        WHILE (pri_lst(p).c_pri /= curnt) LOOP
            b4:= p;
            p:= pri_lst(p).nxt;
        END LOOP;
        temp1:= p;
        pri_lst(b4).nxt:= pri_lst(temp1).nxt;
        pri_lst(temp1).nxt:= 0;

        -- insert the node at the new priority
        b4:= 1;
        p:= 1;
        WHILE (pri_lst(p).c_pri /= new_p) LOOP
            b4:= p;
            p:= pri_lst(p).nxt;
        END LOOP;
        temp2:= p;
        pri_lst(b4).nxt:= temp1;
        pri_lst(temp1).nxt:= temp2;

        -- reset all current priority values
        temp1:= 0;
        p:= 1;
        WHILE (p /= 0) LOOP
            temp1:= temp1 + 1;
            pri_lst(p).c_pri:= temp1;
            p:= pri_lst(p).nxt;
        END LOOP;
        END IF;

    END ripl;
```

```
-- This module enhances the priorities of the radar events as listed in
-- the Priority List if certian conditions are met. The conditions are:
--    1.  The enhance flag must be set to true
--    2.  The ltx value must be greater than the allwd_ltx value


    PROCEDURE enhance(elapsed_time: IN INTEGER) IS

        p: INTEGER:= 1;
        new_pri: INTEGER;

    BEGIN

    -- reset the value of ltx for all events

        WHILE (pri_lst(p).nxt /= 0) LOOP
            IF pri_lst(p).slct_flg THEN
                pri_lst(p).ltx:= 0;
                pri_lst(p).slct_flg:= false;
            ELSE
                pri_lst(p).ltx:= pri_lst(p).ltx + elapsed_time;
            END IF;
            p:= pri_lst(p).nxt;
        END LOOP;

    -- traverse the priority list
        FOR p IN lo_enhnc..max_pri LOOP

        -- look only for events which can be enhanced
            IF ((pri_lst(p).enhnc) AND (pri_lst(p).status)) THEN

            -- is elapsed time more than allowed time?
                IF (pri_lst(p).ltx > pri_lst(p).allwd_ltx) THEN

                -- current priority is above standard enhancement value
                -- but below the lowest enhancement value
                    IF ((pri_lst(p).c_pri <= (pri_lst(p).b_pri - 4)) AND
                        (pri_lst(p).c_pri > lo_enhnc)) THEN

                        new_pri:= pri_lst(p).c_pri - 1;
                    ELSE
                        new_pri:= pri_lst(p).b_pri - 4;
                        -- do not enhance past the lowest enhancement value
                        IF (new_pri < lo_enhnc) THEN
                            new_pri:= lo_enhnc;
                        END IF;
                    END IF;

                -- insert the event at its new priority in the Radar Event
                -- Priority List
                    ripl(pri_lst(p).c_pri, new_pri);   -- from rsm3a
                END IF;
            END IF;
        END LOOP;
    END enhance;

END rsm3;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 8 Dec 86
-- MODULE TYPE: Radar Scheduler Subroutines
-- PRUPOSE: Support the Beam Selection process
-- NAME: Beam Selection Routines... RSM5.LIB  vers 2.0

-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 30 Aug 86
-- MODULE TYPE: Radar Scheduler Subroutine
-- PRUPOSE: insert a node at the end of a linked list
-- NAME: llend ... RSM5A

-- This subroutine has two input parameters: "q" is a pointer to the
-- node which is to be inserted. "s" is a pointer to  the list to which
-- the node is to be inserted at the end of.

-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 30 Aug 86
-- MODULE TYPE: Beam selection subroutine
-- PURPOSE: assign a Radar Event Control Node from a pool of available
-- nodes
-- NAME: Get Rect Node ... RSM5B

-- This module locates the first available RECT node from the pool. It
-- removes the node from the pool and passes its pointer back to the
-- invoking procedure.

-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 30 Aug 86
-- MODULE TYPE: Beam Selection subroutine
-- PURPOSE: return an unused RECT node to the available pool of nodes
-- NAME: Free RECT node ... RSM5C

-- This module returns an unused radar eevent control node to the
-- available pool of radar event control nodes.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH rsm0;
PACKAGE rsm5 IS

    USE rsm0;
    PROCEDURE llend(q,s: IN OUT RECTPtr);

    FUNCTION get_rect_node RETURN RECTPtr;

    PROCEDURE free_rect_node(p: IN OUT RECTPtr);

END rsm5;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 8 Dec 86
-- MODULE TYPE: Radar Scheduler Subroutines  vers 2.0
-- PRUPOSE: Support the Beam Selection process
-- NAME: Beam Selection Routines... RSM5.PKG


    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH tab1,tab7,rsm0;
PACKAGE BODY rsm5 IS

    USE tab1,tab7,rsm0;

    -- pointers for procedure/function use only
    p: RECTPtr;

-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 30 Aug 86
-- MODULE TYPE: Radar Scheduler Subroutine
-- PRUPOSE: insert a node at the end of a linked list
-- NAME: llend ... RSM5A

-- This subroutine has two input parameters: "q" is a pointer to the
-- node which is to be inserted. "s" is a pointer to  the list to which
-- the node is to be inserted at the end of.

    PROCEDURE llend(q,s: IN OUT RECTPtr) IS

    BEGIN
        -- set the new node's pointer to null
        q.nxt_event:= null;

        -- check for an empty list
        IF (s = null) THEN
            s:= q;
        ELSE
            p:= s;
            WHILE (p.nxt_event /= null) LOOP
                p:= p.nxt_event;
            END LOOP;

            -- insert the new node at the end of the list
            p.nxt_event:= q;
        END IF;

    END llend;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 26 Nov 86
-- MODULE TYPE: Beam selection subroutine
-- PURPOSE: assign a Radar Event Control Node from a pool of available
-- nodes
-- NAME: Get Rect Node ... RSM5B

-- This module locates the first available RECT node from the pool. It
-- removes the node from the pool and passes its pointer back to the
-- invoking procedure.


    FUNCTION get_rect_node RETURN RECTPtr IS

    BEGIN
        IF (pool_ptr = null) THEN
            pool_ptr:= NEW RadEveConTab;
            pool_ptr.srch_dwl:= NEW SrchData;
            pool_ptr.trk_dwl:= NEW TrkData;
        END IF;

    -- set p to pool_ptr then relink the list
        p:= pool_ptr;
        pool_ptr:= pool_ptr.nxt_event;
        p.nxt_event:= null;
        RETURN p;
    END get_rect_node;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 2 Dec 86
-- MODULE TYPE: Beam Selection subroutine
-- PURPOSE: return an unused RECT node to the available pool of nodes
-- NAME: Free RECT node ... RSM5C

-- This module returns an unused radar event control node to the
-- available pool of radar event control nodes.


    PROCEDURE free_rect_node(q: IN OUT RECTPtr) IS

    BEGIN
        -- set link to null
        q.nxt_event:= null;

        -- insert the node at the end of the node pool list
        IF (pool_ptr = null) THEN
            pool_ptr:= q;
        ELSE
            p:= pool_ptr;
            WHILE (p.nxt_event /= null) LOOP
                p:= p.nxt_event;
            END LOOP;
            -- insert unused node
            p.nxt_event:= q;
        END IF;

    END free_rect_node;

BEGIN

    -- initialize pointer one time to avoid reinitialzation every
    -- time procedure/function is called.
    p:= NEW RadEveConTab;

END rsm5;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 30 Sep 86
-- MODULE TYPE: internal radar scheduler routine  vers 2.0
-- PURPOSE: ensure dwells satisfy the hardware constraints
-- NAME: hw_constraint ... RSM9.LIB

-- For test purposes this routine assigns a fixed percentage of
-- the Radar Resources available to the current beam being formatted.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH tab0;
PACKAGE rsm9 IS

    USE tab0;
    PROCEDURE hw_constraint(id:IN QueType; rru,dru:IN OUT INTEGER;
                                hwc:OUT BOOLEAN);

END rsm9;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 30 Sep 86
-- MODULE TYPE: internal radar scheduler routine  vers 2.0
-- PURPOSE: ensure dwells satisfy the hardware constraints
-- NAME: hw_constraint ... RSM9.PKG

-- For test purposes this routine assigns a fixed percentage of
-- the Radar Resources available to the current beam being formatted.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH tab0;

PACKAGE BODY rsm9 IS

    USE tab0;
    PROCEDURE hw_constraint(id:IN QueType; rru,dru:IN OUT INTEGER;
                                    hwc:OUT BOOLEAN) IS

        srch_pcnt: CONSTANT INTEGER:= 3;
        sr_pcnt: CONSTANT INTEGER:= 6;
        trk_pcnt: CONSTANT INTEGER:= 7;
        mssl_pcnt: CONSTANT INTEGER:= 7;

        percent: INTEGER;

    BEGIN
        -- determine correct percent of resource for type queue

        CASE id IS
            WHEN search => percent:= srch_pcnt;
            WHEN special_request => percent:= sr_pcnt;
            WHEN track => percent:= trk_pcnt;
            WHEN missile => percent:= mssl_pcnt;
        END CASE;

        rru:= rru + percent;

        -- are the hardware constraints satisfied ?

        IF (rru < 100) THEN
            dru:= 100 - rru;
            hwc:= true;
        ELSE
            rru:= rru - percent;
            hwc:= false;
        END IF;

    END hw_constraint;

END rsm9;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 30 Sep 86
-- MODULE TYPE: internal radar scheduler routine  vers 2.0
-- PURPOSE: output selected and formatted dwells
-- NAME: fill external tables ... RSM10.LIB

-- This module fills the two common memory interface dwell buffers
-- with the data on the formatted dwells. The buffers are double
-- buffered circularly linked lists. Each time this module is executed
-- the pointers are advanced to the next node in the list.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH tab32,tab40,rsm0;

PACKAGE rsm10 IS

    USE tab32,tab40,rsm0;

    PROCEDURE fill_ext_tab(p1,p2:IN OUT PtrOccb;p3,p4:IN OUT PtrRto0;
                                pr:IN OcbData);

END rsm10;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 30 Sep 86
-- MODULE TYPE: internal radar scheduler routine  vers 2.0
-- PURPOSE: output selected and formatted dwells
-- NAME: fill external tables ... RSM10.PKG

-- This module fills the two common memory interface dwell buffers
-- with the data on the formatted dwells. The buffers are double
-- buffered circularly linked lists. Each time this module is executed
-- the pointers are advanced to the next node in the list.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH tab32,tab40,rsm0;

PACKAGE BODY rsm10 IS

    USE tab32,tab40,rsm0;

    PROCEDURE fill_ext_tab(p1,p2:IN OUT PtrOccb;p3,p4:IN OUT PtrRtoO;
                           pr:IN OcbData) IS


    BEGIN
        IF ((p1 /= p2) AND (p3 /= p4)) THEN
            p1:= p1.link;
            p3:= p3.link;
        END IF;

        -- fill channel buffer structure

        p1.oa.cntrl_word.rdr_xmsn_on:= pr.xmit_flg;
        p1.om.face:= pr.face_assign;
        p1.oh.pri1_mti:= pr.pri_mti(1);
        p1.oh.pri2_mti:= pr.pri_mti(2);
        p1.ot(1).otlsb:= pr.mis_up_com;
        p1.ot(1).otmsb:= pr.mis_indx;
        p1.ol.xmit_freq:= pr.xmit_freq_chnl;
        p1.ol.rcv_freq:= pr.rcv_freq_chnl;
        p1.oj.subchnl_freq_group:= pr.subchnl_freq_grp;
        p1.o_f.phse1_code:= pr.phse_code;
        p1.og.fdbk1:= pr.fdbk_phsecode;
        p1.oa.cntrl_word.freq_group_slct:= pr.freq_grp_slct;
        p1.oi.dwl_1_start_time:= pr.detect_thrslds(1);
        p1.ob.detect1_thrsld:= pr.detect_thrslds(1);
        p1.ob.detect2_thrsld:= pr.detect_thrslds(2);
        p1.ob.detect3_thrsld:= pr.detect_thrslds(3);
        p1.oe.trunc1_thrsld:= pr.trunc_thrslds(1);
        p1.oe.trunc2_thrsld:= pr.trunc_thrslds(2);
        p1.oq.elev_sector:= pr.elev_sctr;
        p1.os.dply_azim:= pr.dply_azim;
        p1.o_r.dply_elev:= pr.dply_elev;
        p1.os.video_extnt:= pr.rge_trk_gte_strt;

        -- fill R_to_O Table structure

        p3.dwl_data.mode:= pr.mjr_a_mode;
        p3.dwl_data.face:= pr.face_assign;
        p3.dwl_data.sub_mode(1):= pr.b_submode;
        p3.dwl_data.sub_mode(2):= pr.c_submode;
        p3.dwl_data.dwl_idx:= pr.dwl_idx_num;
        p3.dwl_data.beam_purpose:= pr.ocb_purp;
        p3.dwl_data.dwl_start_idx:= pr.dwl_strt_ctl;
        p3.dwl_data.doct_unblnk_gates:= pr.doct_blnk_gte;
        p3.dwl_data.clutter_unblnk_gates:= pr.cltr_blnk_gte;
```

```
    END fill_ext_tab;
END rsm10;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 21 Aug 86
-- MODULE TYPE: internal Radar Scheduler routine  vers 2.0
-- PURPOSE: compute scheduling interval elapsed time
-- NAME: Elapsed Time ... RSM12.LIB

-- This module is designed to compute the amount of time the Radar
-- Scheduler has spent selecting a beam from the current requested
-- event queue. The routine swaps the old real time value for the
-- current real time value (in milliseconds) and then computes the
-- new value of the real time  and updates the elapsed time.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

PACKAGE rsm12 IS

    PROCEDURE elapsed_time(et,rtim: IN OUT INTEGER);

END rsm12;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 21 Aug 86
-- MODULE TYPE: internal Radar Scheduler routine
-- PURPOSE: compute scheduling interval elapsed time  vers 2.0
-- NAME: Elapsed Time ... RSM12.PKG

-- This module is designed to compute the amount of time the Radar
-- Scheduler has spent selecting a beam from the current requested
-- event queue. The routine swaps the old real time value for the
-- current real time value (in milliseconds) and then computes the
-- new value of the real time  and updates the elapsed time.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH global;

PACKAGE BODY rsm12 IS

    USE global;

    PROCEDURE elapsed_time(et,rtim: IN OUT INTEGER) IS

        oltim: INTEGER;

    BEGIN

        oltim:= rtim;
        rtim:= clock();       -- clock from csr7 in global
        et:= et + rtim - oltim;

    END elapsed_time;

END rsm12;
```

```
-- OWNER: AEGIS MODELLING GROUP
-- DATE OF LAST UPDATE: 30 Sep 86
-- MODULE TYPE: Internal Radar Scheduler routine  vers 2.0
-- PURPOSE: Print out the results of the latest scheduling interval
--          for analysis
-- NAME: Dump ... RSM13.LIB

-- This routine prints out the information on the radar request queues
-- and the scheduled dwells for the current interval. The information
-- printed consists of:
--
--   1.  Requested Beam Listing
--       a.  The name of the Event whose queue is being printed
--       b.  The identification code of the requested beam
--       c.  The requested beam's position within the queue
--
--   2.  Scheduled Dwell Listing
--       a.  The scheduled dwell's unique identification code
--       b.  The amount of Radar Resources remaining after the
--           scheduling of the dwell - expressed as a percentage
--       c.  The index into the current interval's Radar Event
--           Control Table

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH Io;

PACKAGE rsm13 IS

    USE Io;
    Text: file;

    PROCEDURE dump;

END rsm13;
```

```
-- OWNER: AEGIS MODELLING GROUP
-- DATE OF LAST UPDATE: 26 Nov 86
-- MODULE TYPE: Internal Radar Scheduler routine  vers 2.0
-- PURPOSE: Print out the results of the latest scheduling interval
--          for analysis
-- NAME: Dump ... RSM13.PKG

-- This routine prints out the information on the radar request queues
-- and the scheduled dwells for the current interval. The information
-- printed consists of:
--
--   1.  Requested Beam Listing
--       a.  The name of the Event whose queue is being printed
--       b.  The identification code of the requested beam
--       c.  The requested beam's position within the queue
--
--   2.  Scheduled Dwell Listing
--       a.  The scheduled dwell's unique identification code
--       b.  The amount of Radar Resources remaining after the
--       c.  The index to the current interval's Radar Event
--           Control Table

   pragma arithcheck(off);    pragma debug(off);
   pragma enumtab(off);       pragma rangecheck(off);
@  pragma arithcheck(off);    pragma debug(off);
@  pragma enumtab(off);       pragma rangecheck(off);

WITH tab0,tab1,tab2,rsm0,Io,Util;

PACKAGE BODY rsm13 IS

   USE tab0,tab1,tab2,rsm0,Io,Util;

   dsh: string(55):="-------------------------------------------------------";
   sptr: SearchPtr;
   tptr: TrkPtr;

   PROCEDURE dump IS

      ctr: INTEGER:= 1;
      start,ctn,i: INTEGER;

   BEGIN

      Put(Text,REQUESTED BEAMS FOR SCHEDULER INTERVAL: ");
      Put(Text,intvl_num,5); New_Line(Text);
      Put(Text,dsh); New_Line(Text);
      WHILE (ctr /= 0) LOOP
         Put(Text,pri_lst(ctr).eventnm); New_Line(Text);

         IF pri_lst(ctr).status THEN
            Put(Text,"BEAM ID       ");
            i:= 0;

            CASE pri_lst(ctr).que_id IS

               WHEN search | special_request =>
                  sptr:= pri_lst(ctr).que_ptr.Snode;
                  WHILE ((i < 10) AND (sptr /= null)) LOOP
                     i:= i + 1;
                     Put(Text,sptr.info .bid); Put(Text," ");
                     sptr:= sptr.nxt;
                  END LOOP;

               WHEN track | missile =>
                  tptr:= pri_lst(ctr).que_ptr.Tnode;
                  WHILE ((i < 10) AND (tptr /= null)) LOOP
                     i:= i + 1;
                     Put(Text,tptr.info .bid); Put(Text," ");
```

```
                    tptr:= tptr.nxt;
                END LOOP;

            END CASE;

            New_Line(Text);
            Put(Text,"QUEUE POSIT  ");
            FOR ctn IN 1..i LOOP
                Put(Text,ctn,4);
            END LOOP;

            New_Line(Text);
            Put(Text,dsh); New_Line(Text);

        ELSE

            Put(Text,"           NO REQUESTS THIS INTERVAL");
            New_Line(Text); Put(Text,dsh); New_Line(Text);
        END IF;

        ctr:= pri_lst(ctr).nxt;

    END LOOP;

    New_Line(Text);
    Put(Text,"SCHEDULED DWELLS FOR SCHEDULER INTERVAL: ");
    Put(Text,intvl_num,5); New_Line(Text);
    Put(Text,dsh); New_Line(Text);
    rptr:= rect_ptr;
    start:= 1;

    WHILE (rptr /= null) LOOP
        p:= rptr;
        i:= 0;
        Put(Text,"BEAM ID       ");
        WHILE ((i < 10) AND (p /= null)) LOOP
            i:= i + 1;
            Put(Text,p.beamid); Put(Text," ");
            p:= p.nxt_event;
        END LOOP;
        New_Line(Text);
        Put(Text,"RESOURCES    ");
        i:= 0;
        p:= rptr;
        WHILE ((i < 10) AND (p /= null)) LOOP
            i:= i + 1;
            Put(Text,p.dru,4);
            p:= p.nxt_event;
        END LOOP;
        New_Line(Text);
        Put(Text,"DWELL #      ");
        FOR ctn IN start..(start + i - 1) LOOP
            Put(Text,ctn,4);
        END LOOP;
        New_Line(Text);
        Put(Text,dsh); New_Line(Text);
        IF (p /= null) THEN
            rptr:= p;
            start:= start + i;
        ELSE
            rptr:= null;
        END IF;

    END LOOP;
    Put(Text,dsh); New_Line(Text); New_Line(Text);

END dump;

BEGIN
```

```
    -- initialize working pointers one time to avoid reinitialization
    -- each time procedure is invoked.
    sptr:= NEW SearchNode;
    tptr:= NEW TrackNode;
    rptr:= NEW RadEveConTab;
    p:= NEW RadEveConTab;

END rsm13;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 29 Aug 86
-- MODULE TYPE: internal Radar Scheduler routine  vers 2.0
-- PURPOSE: free memory for the next scheduling interval
-- NAME: Free Memory ... RSM14.LIB

-- This module traverses the available RECT pool list and inserts the
-- current Radar Event Control Table list at the end of the pool.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH rsm0;

PACKAGE rsm14 IS

    USE rsm0;

    PROCEDURE free_memory(pool_ptr,rect_ptr:IN OUT RECTPtr);

END rsm14;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 29 Aug 86
-- MODULE TYPE: internal Radar Scheduler routine  vers 2.0
-- PURPOSE: free memory for the next scheduling interval
-- NAME: Free Memory ... RSM14.PKG

-- This module traverses the available RECT pool list and inserts the
-- current Radar Event Control Table list at the end of the pool.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH rsm0;

PACKAGE BODY rsm14 IS

    USE rsm0;

    -- pointer for procedure use only
    p: RECTPtr;

    PROCEDURE free_memory(pool_ptr,rect_ptr:IN OUT RECTPtr) IS

    BEGIN
        IF (pool_ptr = null) THEN
            pool_ptr:= rect_ptr;
            rect_ptr:= null;
        ELSE
            p:= pool_ptr;
            WHILE (p.nxt_event /= null) LOOP
                p:= p.nxt_event;
            END LOOP;
            p.nxt_event:= rect_ptr;
            rect_ptr:= null;
        END IF;
    END free_memory;

BEGIN

    -- initialize working pointer one time to avoid
    -- reinitialization each time procedure is invoked.
    p:= NEW RadEveConTab;

END rsm14;
```

APPENDIX D

TEST HARNESS SOURCE CODE

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 24 Sep 86
-- MODULE TYPE: Display Subroutine  vers 2.0
-- PURPOSE: Operator Interface
-- NAME: Operator Interface Module ... SADM1.LIB

-- This module provides the user of the Radar Scheduler Test
-- Harness (SPY.COM) with the ability to alter some of the
-- major program parameters prior to execution of the program
-- with out having to alter the source code, recompile, and
-- link the program.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

PACKAGE sadm1 IS

    PROCEDURE oim(numtrks,numintvls,skipintvls: OUT INTEGER);

END sadm1;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 24 Sep 86
-- MODULE TYPE: Display Subroutine  vers 2.0
-- PURPOSE: Operator Interface
-- NAME: Operator Interface Module ... SADM1.PKG

-- This module provides the user of the Radar Scheduler Test
-- Harness (SPY.COM) with the ability to alter some of the
-- major program parameters prior to execution of the program
-- with out having to alter the source code, recompile, and
-- link the program.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH Util;

PACKAGE BODY sadm1 IS

    USE Util;

    PROCEDURE oim(numtrks,numintvls,skipintvls: OUT INTEGER) IS

        astrk15: CONSTANT STRING:= "***************";
        space10: CONSTANT STRING:= "          ";

    BEGIN
        Put(astrk15); Put("       AEGIS AN/SPY1-A       ");
        Put(astrk15); New_Line;
        Put(astrk15); Put("   RADAR SCHEDULER PROGRAM   ");
        Put(astrk15); New_Line;
        Put(astrk15); Put(astrk15); Put(astrk15); Put(astrk15);
        New_Line; New_Line; New_Line; New_Line;
        Put("Input the number of tracks to be initialized.");
        New_Line; New_Line; Put(space10);
        Put("NUMBER OF TRACKS ---> "); Get(numtrks); New_Line;
        New_Line; New_Line;
        Put("Input the number of scheduling intervals to be run.");
        New_Line; New_Line; Put(space10);
        Put("NUMBER OF INTERVALS ---> "); Get(numintvls);
        New_Line; New_Line; New_Line;
        Put("Define the interval display delay period.");
        New_Line; New_Line; Put(space10);
        Put("SKIP INTERVALS ---> "); Get(skipintvls); New_Line;
        New_Line; New_Line; New_Line;
        Put(space10); Put(space10); Put("BEGIN EXECUTION");
        New_Line;
    END oim;

END sadm1;
```

```
-- OWNER: AEGIS MODELING GROUP
-- DATE OF LAST UPDATE : 31 AUG 86
-- MODULE TYPE: PROCESS  VERS 3.0
-- PURPOSE: MODEL THE RADAR SCHEDULER FUNCTION
-- NAME: SEARCH MANAGEMENT ... SRCM.LIB


-- The purpose of this module is to manage the
-- request of search and special request radar
-- events.  The current design processes static
-- data for the Radar Scheduler Test Harness.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

PACKAGE srcm IS

    PROCEDURE searchmgmt;

END srcm;
```

```
-- OWNER: AEGIS MODELING GROUP
-- DATE OF LAST UPDATE : 30 Sep 86
-- MODULE TYPE: PROCESS  VERS 3.0
-- PURPOSE: MODEL THE RADAR SCHEDULER FUNCTION
-- NAME: SEARCH MANAGEMENT ... SRCM.PKG


-- The purpose of this module is to manage the
-- request of search and special request radar
-- events.  The current design processes static
-- data for the Radar Scheduler Test Harness.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH global,tab0,smm1,smm2;

PACKAGE BODY srcm IS

    USE global,tab0,smm1,smm2;

    PROCEDURE searchmgmt IS

        lquad: INTEGER;
        TYPE BeamCount IS ARRAY(1..2) OF INTEGER;
        bm_ctr: BeamCount;
        ppl,rnum,i: INTEGER;

    BEGIN

        -- not initialized for test purposes, taken from smm1
        -- lquad:= 1;
        -- bm_ctr(1):= 0;
        -- bm_ctr(2):= 0;

        sm_init;                          -- from smm1.pkg
        FOR i IN 1..infinity LOOP
            await(s_pid,es_id,i);

            -- traverse the radar event priority list
            ppl:=1;
            WHILE (ppl /= 0) LOOP
                IF ((pri_lst(ppl).que_id = search) OR
                     (pri_lst(ppl).que_id = special_request)) THEN

                    fill_que(pri_lst(ppl));

                END IF;
                -- point to the nest priority in the list
                ppl:= pri_lst(ppl).nxt;
            END LOOP;
        END LOOP;
    END searchmgmt;

END srcm;
```

```
-- AEGIS MODELING GROUP
-- DATE OF LAST UPDATE: 31 AUG 86
-- MODULE TYPE: SEARCH MANAGEMENT MODULE  vers 2.0
-- PURPOSE: INITIALIZE THE SEARCH EVENT QUEUES IN THE
--          PRIORITY LIST
-- NAME: SEARCH MANAGEMENT INITIALIZATION ... SMM1.LIB

-- This module is executed once.  Its purpose is to allocate
-- memory for search nodes (Table 1) and create empty request
-- queues for each search and special request radar event in
-- the Radar Event Priority List (Table 0).

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

PACKAGE smm1 IS

    PROCEDURE sm_init;

END smm1;
```

```
-- AEGIS MODELING GROUP
-- DATE OF LAST UPDATE: 26 Nov 86
-- MODULE TYPE: SEARCH MANAGEMENT MODULE  vers 2.0
-- PURPOSE: INITIALIZE THE SEARCH EVENT QUEUES IN THE
--          PRIORITY LIST
-- NAME: SEARCH MANAGEMENT INITIALIZATION ... SMM1.PKG

-- This module is executed once.  Its purpose is to allocate
-- memory for search nodes (Table 1) and create empty request
-- queues for each search and special request radar event in
-- the Radar Event Priority List (Table 0).

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH tab0,tab1;

PACKAGE BODY smm1 IS

    USE tab0,tab1;

    PROCEDURE sm_init IS

        p,qptr: SearchPtr:= NEW SearchNode;
        q: SearchPtr;
        ctr: INTEGER:= 1;
        node_ctr: INTEGER;

    BEGIN

        -- traverse the Priority Event List
        WHILE (ctr /= 0) LOOP
            IF ((pri_lst(ctr).que_id = search) OR
                (pri_lst(ctr).que_id = special_request)) THEN

                -- initialize the queue to length - max_nodes

                p:= pri_lst(ctr).que_ptr.Snode;
                node_ctr:= 0;
                WHILE (node_ctr < pri_lst(ctr).max_nodes) LOOP
                    node_ctr:= node_ctr + 1;
                    q:= NEW SearchNode;
                    q.info:= NEW SrchData;
                    q.nxt:= null;
                    -- insert at the end of event queue p
                    IF (p = null) THEN
                        p:= q;
                        pri_lst(ctr).que_ptr.Snode:= q;
                    ELSE
                        -- search for the end of the event queue
                        qptr:= p;
                        WHILE (qptr.nxt /= null) LOOP
                            qptr:= qptr.nxt;
                        END LOOP;
                        -- insert the new node
                        qptr.nxt:= q;
                    END IF;
                END LOOP;
            END IF;
            ctr:= pri_lst(ctr).nxt;
        END LOOP;

    END sm_init;

END smm1;
```

```
-- OWNER: AEGIS MODELING GROUP
-- DATE OF LAST UPDATE: 30 Sep 86
-- MODULE TYPE: SEARCH MANAGEMENT SUBROUTINE  vers 2.0
-- PURPOSE: FILL A PRIORITY LIST SEARCH QUEUE
-- NAME: FILL SEARCH QUEUE ...SMM2.LIB

-- This routine is responsible for filling a priority
-- event queue with the proper synchronous beam request
-- data.  It executes the following functions for each
-- event in the priority list which corresponds to a
-- Search Management event.  The Fill Search Queue
-- subroutine calculates; beam mode, unique beam id,
-- beam position (azimuth and elevation), instrumented
-- range, blanking gates, the end of frame indicator,
-- the Radar Event Priority List (Table 0).-- and requestor identity.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH tab0;

PACKAGE smm2 IS

    USE tab0;

    PROCEDURE fill_que(pri_lst: IN OUT PriLstPtr);

    TYPE BeamCtrArray IS ARRAY(1..2) OF INTEGER;
    bm_ctr: BeamCtrArray;

    lquad: INTEGER;

END smm2;
```

```
-- OWNER: AEGIS MODELING GROUP
-- DATE OF LAST UPDATE: 30 Sep 86
-- MODULE TYPE: SEARCH MANAGEMENT SUBROUTINE  vers 2.0
-- PURPOSE: FILL A PRIORITY LIST SEARCH QUEUE
-- NAME: FILL SEARCH QUEUE ...SMM2.PKG

-- This routine is responsible for filling a priority
-- event queue with the proper synchronous beam request
-- data.  It executes the following functions for each
-- event in the priority list which corresponds to a
-- Search Management event.  The Fill Search Queue
-- subroutine calculates; beam mode, unique beam id,
-- beam position (azimuth and elevation), instrumented
-- range, blanking gates, the end of frame indicator,
-- the Radar Event Priority List (Table 0).-- and requestor identity.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH global,tab0,tab1,StrLib;

PACKAGE BODY smm2 IS

    USE global,tab0,tab1,StrLib;

-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 1 Oct 86
-- MODULE TYPE: Search Management subroutine
-- PURPOSE: Calculate beam position
-- NAME: Beam Position ... SMM2A

-- This subroutine calculates the beam position for search type
-- beams based on the calling parameters - que identity and random
-- number, the last quadrent from which the beam was requested
-- and a pointer to the event node for which the beam position is
-- being calculated.


    PROCEDURE bm_pos(qid: IN QueType;rnum: IN INTEGER; quad: IN OUT INTEGER;
                     p: IN OUT SearchPtr) IS

    BEGIN
        IF (quad = 1) THEN
            quad:= 3;
        ELSIF (quad = 2) THEN
            quad:= 4;
        ELSIF (quad = 3) THEN
            quad:= 2;
        ELSIF (quad = 4) THEN
            quad:= 1;
        END IF;

    -- the rest of this module has not been implemented yet

    END bm_pos;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 1 Oct 86
-- MODULE TYPE: Search Management subroutine
-- PURPOSE: Calculate end of frame for search type beams
-- NAME: End Frame ... SMM2B

-- This subroutine sets the end of frame indication flag based on
-- its input parameters - beams requested and event identity. An
-- end of search frame is indicated for the horizon search event
-- after 600 beams have been scheduled. An end of search frame is
-- indicated for the above horizon search after 3600 beams have
-- been scheduled.


    FUNCTION end_frm(num_bms,eid: IN INTEGER) RETURN BOOLEAN IS

    BEGIN
        IF ((eid = 9) AND (num_bms >= 600)) THEN
            RETURN true;
        ELSIF ((eid = 22) AND (num_bms >= 3600)) THEN
            RETURN true;
        ELSE
            RETURN false;
        END IF;
    END end_frm;
```

```
PROCEDURE fill_que(pri_lst: IN OUT PriLstPtr) IS

    nodes_filld: INTEGER:= 0;
    temp: STRING(10);
    qptr,p: SearchPtr:= NEW SearchNode;
    rnum: INTEGER;

BEGIN

    -- set event que status
    rnum:= rand();
    IF ((pri_lst.que_id = special_request) AND
         ((rnum MOD max_pri) /= 0)) THEN

       pri_lst.status:= false;
    ELSE
       pri_lst.status:= true;
    END IF;

    pri_lst.que_ptr.Snode.info:= NEW SrchData;
    p.info:= NEW SrchData;
    qptr.info:= NEW SrchData;
    -- set queue pointer to event(trip) pointer
    qptr:= pri_lst.que_ptr.Snode;

    -- traverse the event queue and fill data fields
    p:= qptr;
    WHILE ((p /= null) AND (pri_lst.status)) LOOP

        -- increment the number of nodes filled to maintain unique
        -- beam id. set mode to event base pri
        nodes_filld:= nodes_filld + 1;
        p.info.mode:= pri_lst.b_pri;

        -- assign unique id code to this beam
        p.info.bid:= Extract(pri_lst.eventnm,1,1);  -- from StrLib
        temp:= Int_to_Str(nodes_filld);   -- from StrLib
        IF nodes_filld < 10 THEN
            temp:= Insert("0",temp,1);
        END IF;
        p.info.bid:= Insert(temp,p.info.bid,2);

        -- calculate beam position - 1) queue id #, 2) random #,
        -- 3) last quadrant entered, 4) pointer p
        rnum:= rand();
        bm_pos(pri_lst.que_id,rnum,lquad,p);

        -- calculate the instrumented range
        -- calculate blanking gates
        -- not implemented in this version

        -- calculate end of frame
        IF (pri_lst.que_id = search) THEN
            IF (pri_lst.b_pri = 9) THEN    -- horizon search frame
                bm_ctr(1):= bm_ctr(1) + 1;
                p.info.eof_indic:= end_frm(bm_ctr(1),9);
            ELSE     -- above horizon search frame
                bm_ctr(2):= bm_ctr(2) + 1;
                p.info.eof_indic:= end_frm(bm_ctr(2),22);
            END IF;
        ELSE       -- is a special request event
            p.info.eof_indic:= false;
        END IF;

        -- assign requestor id. for this version the requestor id
        -- is assigned the value of the beam mode.
        p.info.req_id:= p.info.mode;

        -- point to the next node in the event queue
```

```
            p:= p.nxt;
        END LOOP;
    END fill_que;
END smm2;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 29 Sep 86
-- MODULE TYPE: Process  vers 3.0
-- PURPOSE: Model the Radar Scheduler function
-- NAME: Detection Processing ... DRCM.LIB

-- This process models the Detection Processing process for the
-- purpose of modeling its interface to the Radar Scheduler process.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

PACKAGE drcm IS

PROCEDURE detectproc;

END drcm;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 2 Dec 86
-- MODULE TYPE: Process  vers 3.0
-- PURPOSE: Model the Radar Scheduler function
-- NAME: Detection Processing ... DRCM.PKG

-- This process models the Detection Processing process for the
-- purpose of modeling its interface to the Radar Scheduler process.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH global,tab7,dpm1;

PACKAGE BODY drcm IS

    USE global,tab7,dpm1;

    PROCEDURE detectproc IS

        ctr,i: INTEGER;

    BEGIN
        -- initialize the Track File
        trkfilinit(ptrk);    -- See DPM1.PKG

        FOR i IN 1..infinity LOOP
            await(d_pid,ed_id,i);
            FOR ctr IN 1..126 LOOP
                null;
            END LOOP;
        END LOOP;
    END detectproc;

END drcm;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 1 Oct 86
-- MODULE TYPE: Detection Processing routine  vers 2.0
-- PURPOSE: Initialize the Track File
-- NAME: Track File Initialization ... DPM1.LIB

-- This module creates the Track File (table 7) used by the test
-- harness to provide the Radar Scheduler with viable track data.
-- This subroutine invokes the common service routine - rand and
-- the Detection Processing subroutine dpinend.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH tab7;

PACKAGE dpm1 IS

    USE tab7;

    PROCEDURE trkfilinit(ptrk:OUT PtrTrkFile);

END dpm1;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 2 Dec 86
-- MODULE TYPE: Detection Processing routine  vers 2.0
-- PURPOSE: Initialize the Track File
-- NAME: Track File Initialization ... DPM1.PKG

-- This module creates the Track File (table 7) used by the test
-- harness to provide the Radar Scheduler with viable track data.
-- This subroutine invokes the common service routine - rand and
-- the Detection Processing subroutine dpinend.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH global,tab4,tab7,dpm1a,Longops;

PACKAGE BODY dpm1 IS

    USE global,tab4,tab7,dpm1a,Longops;

    PROCEDURE trkfilinit(ptrk: OUT PtrTrkFile) IS

        p: PtrTrkFile;
        i,j,rnum:INTEGER;

    BEGIN
        ptrk:=null;
        FOR i IN 1..A_to_R.op_doct.mtrks LOOP

            -- create a Track File node
            p:= NEW TrkFile;
            p.beam_data:= NEW TrkData;

            -- initialize the beam data based on a random number
            rnum:= rand();

            -- compute the mode and priority
            j:= (rnum MOD 22);
            IF (((j >= 4) AND (j <= 8)) OR ((j >= 14) AND (j <= 21))) THEN
                p.beam_data.mode:= j;
            ELSE
                p.beam_data.mode:= j + 5;
            END IF;
            p.beam_data.priority:= p.beam_data.mode;

            -- these flags are constant valued
            p.beam_data.xmit_req_flg:= true;
            p.beam_data.sim_tgt_flg:= false;

            -- compute log amplitude estimate of echo signal
            p.beam_data.log_ampld_est:= (rnum MOD 100);

            -- compute "z" position
            p.beam_data.posit.z:= (rnum MOD 1000);
            IF (p.beam_data.posit.z < 1000) THEN
                p.beam_data.low_elev_trk_flg:= true;
            ELSE
                p.beam_data.low_elev_trk_flg:= false;
            END IF;

            -- compute "x" and "y" positions
            IF ((rnum MOD 2) = 0) THEN
                p.beam_data.posit.x:= rnum;
                rnum:= rand();
                p.beam_data.posit.y:= 0 - rnum;
            ELSE
                p.beam_data.posit.y:= rnum;
```

```
            rnum:= rand();
            p.beam_data.posit.x:= 0 - rnum;
        END IF;

        -- compute slant range
        p.beam_data.posit.slnt_rnge:= Labs(Lmul(Lint(p.beam_data.posit.x),
                                          Lint(p.beam_data.posit.y)));

        -- compute velocity vectors
        p.beam_data.posit.x_dot:= (p.beam_data.posit.x MOD (-200));
        p.beam_data.posit.y_dot:= (p.beam_data.posit.y MOD (-200));
        p.beam_data.posit.z_dot:= (p.beam_data.posit.z MOD (-10));
        p.beam_data.posit.rge_dot:=
             L_to_int(Lmod(p.beam_data.posit.slnt_rnge,Lint(-100)));

        -- compute cross gate bin number
        p.beam_data.xgte_bin_num:= (rnum MOD 1000);

        -- assign track numbers
        p.beam_data.ctl_grp_trk_num:= i;
        p.beam_data.ctsl:= i + 100;

        -- compute track transition flag
        IF (i < 5) THEN
            p.beam_data.trk_xsitn_flag:= true;
        ELSE
            p.beam_data.trk_xsitn_flag:= false;
        END IF;

        -- insert the track node at the end of the Track File
        dpinend(p,ptrk);    -- from dpm1a.pkg

    END LOOP;

END trkfilinit;

END dpm1;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 1 Oct 86
-- MODULE TYPE: Detection Processing subroutine  vers 2.0
-- PURPOSE: Insert a node at the end of a linked list
-- NAME: dpinend ... DPM1A.LIB

-- This subroutine has two parameters: new_node is a pointer to the
-- node which is to be inserted, list_head is a pointer to the list
-- to which the node is to be inserted at the end of.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH tab7;

PACKAGE dpm1a IS

    USE tab7;
    PROCEDURE dpinend(new_node,list_head: IN OUT PtrTrkFile);

END dpm1a;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 1 Oct 86
-- MODULE TYPE: Detection Processing subroutine  vers 2.0
-- PURPOSE: Insert a node at the end of a linked list
-- NAME: dpinend ... DPM1A.PKG

-- This subroutine has two parameters: new_node is a pointer to the
-- node which is to be inserted, list_head is a pointer to the list
-- to which the node is to be inserted at the end of.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH tab7;

PACKAGE BODY dpm1a IS

    USE tab7;

    PROCEDURE dpinend(new_node,list_head: IN OUT PtrTrkFile) IS

        tp: PtrTrkFile:= NEW TrkFile;

    BEGIN
        new_node.nxt_trk:= null;

        -- check for an empty list
        IF (list_head = null) THEN
            list_head:= new_node;
        ELSE
            -- search for the end of the list
            tp:= list_head;
            WHILE (tp.nxt_trk /= null) LOOP
                tp:= tp.nxt_trk;
            END LOOP;

            -- insert the new node at the end of the list
            tp.nxt_trk:= new_node;
        END IF;

    END dpinend;

END dpm1a;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 3 Sep 86
-- MODULE TYPE: process  vers 3.0
-- PURPOSE: Model the Radar Scheduler Function
-- NAME: Switch Action And Display Processing ... ARCM.LIB

-- This process is a single thread module which models the Switch
-- Action And Display module on the INTELLEC[tm] MDS system.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

PACKAGE arcm IS

    PROCEDURE swtchactdply;

END arcm;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 3 Oct 86
-- MODULE TYPE: process  vers 3.0
-- PURPOSE: Model the Radar Scheduler Function
-- NAME: Switch Action And Display Processing ... ARCM.PKG

-- This process is a single thread module which models the Switch
-- Action And Display module on the INTELLEC[tm] MDS system.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH sadm1,global,tab4;

PACKAGE BODY arcm IS

    USE sadm1,global,tab4;


    PROCEDURE swtchactdply IS

        ctr,i: INTEGER;

    BEGIN

        -- execute operator interface module (from sadm1.pkg)
        oim(A_to_R.op_doct.mtrks,A_to_R.op_doct.mintvls,
             A_to_R.op_doct.dply_rect);

        -- initialize Radar Event Priority List
        ipl;   -- from global.pkg

        FOR i IN 1..infinity LOOP

            await(a_pid,ea_id,i);
            FOR ctr IN 1..126 LOOP
               null;
            END LOOP;

        END LOOP;

    END swtchactdply;

END arcm;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 3 Sep 86
-- MODULE TYPE: process  vers 2.0
-- PURPOSE: Model the Radar Schedular function
-- NAME: Track Management .. TRCM.LIB

-- The purpose of this module is to manage the request of track and
-- missile radar events. The current design produces static data for
-- the radar scheduler Test Harness.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

PACKAGE trcm IS

    PROCEDURE trackmgmt;

END trcm;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 3 Oct 86
-- MODULE TYPE: process  vers 2.0
-- PURPOSE: Model the Radar Schedular function
-- NAME: Track Management .. TRCM.PKG

-- The purpose of this module is to manage the request of track and
-- missile radar events. The current design produces static data for
-- the radar scheduler Test Harness.

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH global,tab0,tab2,tab7,tmm1,StrLib;

PACKAGE BODY trcm IS

    USE global,tab0,tab2,tab7,tmm1,StrLib;

    PROCEDURE trackmgmt IS

        i,ppl: INTEGER;
        node_ctr: INTEGER;
        temp: STRING(10);
        p: TrkPtr:= NEW TrackNode;
        q: PtrTrkFile:= NEW TrkFile;

    BEGIN
        tm_init;
        FOR i IN 1..infinity LOOP
            -- wait for the radar loop event value
            await(t_pid,et_id,i);
            -- traverse the radar event priority list
            ppl:= 1;
            WHILE (ppl /= 0) LOOP
                IF ((pri_lst(ppl).que_id = track) OR
                    (pri_lst(ppl).que_id = missile)) THEN
                -- traverse the event queue and Track File

                node_ctr:= 0;
                p:= pri_lst(ppl).que_ptr.Tnode;
                q:= ptrk;      -- ptrk from global.lib
                WHILE ((q /= null) AND (p /= null) AND
                        (node_ctr < pri_lst(ppl).max_nodes)) LOOP
                    -- identify this event

                    IF (q.beam_data.mode = pri_lst(ppl).b_pri) THEN
                        node_ctr:= node_ctr + 1;

                        -- set track node mode
                        p.info.mode:= pri_lst(ppl).b_pri;

                        -- set unique beam identifier
                        temp:= Int_to_Str(node_ctr);
                        IF node_ctr < 10 THEN
                            temp:= Insert("0",temp,1);
                        END IF;
                        p.info.bid:= Extract(pri_lst(ppl).eventnm,1,1);
                        p.info.bid:= Insert(temp,p.info.bid,2);

                        p.info.p_trk:= q;
                        p.info.p_trk.beam_data.bid:= p.info.bid;
                        pri_lst(ppl).status:= true;

                        -- reset pointers
                        p:= p.nxt;
                    END IF;
```

```
                    q:= q.nxt_trk;
                END LOOP;
                IF (p /= null) THEN
                    p.info.p_trk:= null;
                END IF;

            END IF;
            ppl:= pri_lst(ppl).nxt;

        END LOOP;   -- end traverse priority list

    END LOOP;   -- end for loop

END trackmgmt;

END trcm;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 3 Sep 86
-- MODULE TYPE: Track Management Module  vers 2.0
-- PURPOSE: Initialize track events in the Radar Event Priority List
-- NAME: Track Management Initialize .. TMM1.LIB

-- The purpose of this module is to allocate memory for track nodes
-- (table 2) and create empty request queues for each search and
-- special request event in the Radar Event Priority List (table 0).

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

PACKAGE tmm1 IS

    PROCEDURE tm_init;

END tmm1;
```

```
-- OWNER: AEGIS Modeling Group
-- DATE OF LAST UPDATE: 30 Nov 86
-- MODULE TYPE: Track Management Module  vers 2.0
-- PURPOSE: Initialize track events in the Radar Event Priority List
-- NAME: Track Management Initialize .. TMM1.PKG

-- The purpose of this module is to allocate memory for track nodes
-- (table 2) and create empty request queues for each search and
-- special request event in the Radar Event Priority List (table 0).

    pragma arithcheck(off); pragma debug(off);
    pragma enumtab(off);    pragma rangecheck(off);
@   pragma arithcheck(on); pragma debug(on);
@   pragma enumtab(on);    pragma rangecheck(on);

WITH tab0,tab2,tab7;

PACKAGE BODY tmm1 IS

    USE tab0,tab2,tab7;

    PROCEDURE tm_init IS

        qptr,p: TrkPtr:= NEW TrackNode;
        q: TrkPtr;
        ppl: INTEGER:= 1;
        node_ctr: INTEGER;

    BEGIN

        -- traverse the Radar Event Priority List
        WHILE (ppl /= 0) LOOP

            IF ((pri_lst(ppl).que_id = track) OR
                (pri_lst(ppl).que_id = missile)) THEN

                p:= pri_lst(ppl).que_ptr.Tnode;
                node_ctr:= 0;

                WHILE (node_ctr < pri_lst(ppl).max_nodes) LOOP

                    node_ctr:= node_ctr + 1;
                    q:= NEW TrackNode;
                    q.info.p_trk:= NEW TrkFile;
                    q.info.p_trk.beam_data:= NEW TrkData;
                    q.nxt:= null;

                    -- insert node at end of event queue
                    IF (p = null) THEN
                        p:= q;
                        pri_lst(ppl).que_ptr.Tnode:= q;
                    ELSE
                        qptr:= p;
                        WHILE (qptr.nxt /= null) LOOP
                            qptr:= qptr.nxt;
                        END LOOP;
                        qptr.nxt:= q;
                    END IF;

                END LOOP;

            END IF;
            ppl:= pri_lst(ppl).nxt;

        END LOOP;

    END tm_init;

END tmm1;
```

APPENDIX E

SAMPLE RADAR SCHEDULER OUTPUT

```
**************************************************
**************      AEGIS AN/SPY-1A       **************
**************  RADAR SCHEDULER PROGRAM  **************
**************************************************

Instructions to the Operator:

Input the number of tracks to be initialized:

             NUMBER OF TRACKS ----> 50

Input the number of scheduling intervals to be run:

             NUMBER OF INTERVALS ----> 100

Define interval display delay period:

             SKIP INTERVALS ----> 10

                        BEGIN EXECUTION

Completed interval: 10
Completed interval: 20
Completed interval: 30
Completed interval: 40
Completed interval: 50
Completed interval: 60
Completed interval: 70
Completed interval: 80
Completed interval: 90
Completed interval: 100
```

```
REQUESTED BEAMS FOR SCHEDULER INTERVAL:    10
-----------------------------------------------------
A-EVENT - ECM BURNTHROUGH
             NO REQUESTS THIS INTERVAL
-----------------------------------------------------
B-EVENT - TARGET DEFINITION
             NO REQUESTS THIS INTERVAL
-----------------------------------------------------
C-EVENT - SPECIAL TEST
             NO REQUESTS THIS INTERVAL
-----------------------------------------------------
G-EVENT - HIGH PRI TRACK TRANSITION
BEAM ID       G01 G02 G03 G04 G05
QUEUE POSIT     1   2   3   4   5
-----------------------------------------------------
I-EVENT - HORIZON SEARCH
BEAM ID       I01 I02 I03 I04 I05 I06 I07 I08 I09 I10
QUEUE POSIT     1   2   3   4   5   6   7   8   9  10
-----------------------------------------------------
F-EVENT - PRE-ENGAGED HOSTILE TARGET
BEAM ID       F01 F02 F03 F04 F05
QUEUE POSIT     1   2   3   4   5
-----------------------------------------------------
E-EVENT - OWN SM-2 MISSILE GUIDANCE
BEAM ID       E01 E02
QUEUE POSIT     1   2
-----------------------------------------------------
D-EVENT - ENGAGED HOSTILE TARGET
BEAM ID       D01 D02 D03 D04 D05
QUEUE POSIT     1   2   3   4   5
-----------------------------------------------------
H-EVENT - HIGH PRI TRACK CONFIRMATION
BEAM ID       H01 H02
QUEUE POSIT     1   2
-----------------------------------------------------
J-EVENT - SPECIAL ECM BURNTHROUGH
             NO REQUESTS THIS INTERVAL
-----------------------------------------------------
K-EVENT - SPECIAL TARGET DEFINITION
             NO REQUESTS THIS INTERVAL
-----------------------------------------------------
L-EVENT - SPECIAL MANUAL SCAN
             NO REQUESTS THIS INTERVAL
-----------------------------------------------------
M-EVENT - SPECIAL TARGET ACQUISITION
             NO REQUESTS THIS INTERVAL
-----------------------------------------------------
N-EVENT - CONFIRMED HOSTILE TRACK
BEAM ID       N01 N02 N03 N04
QUEUE POSIT     1   2   3   4
-----------------------------------------------------
O-EVENT - ASSUMED HOSTILE TRACK
BEAM ID       O01 O02 O03 O04 O05
QUEUE POSIT     1   2   3   4   5
-----------------------------------------------------
P-EVENT - UNEVALUATED TRACK
BEAM ID       P01 P02
QUEUE POSIT     1   2
-----------------------------------------------------
Q-EVENT - CONTROLLED FRIENDLY TRACK
BEAM ID       Q01 Q02 Q03 Q04
QUEUE POSIT     1   2   3   4
-----------------------------------------------------
V-EVENT - ABOVE HORIZON SEARCH
BEAM ID       V01 V02 V03 V04 V05 V06 V07 V08 V09 V10
QUEUE POSIT     1   2   3   4   5   6   7   8   9  10
-----------------------------------------------------
```

```
R-EVENT - TRACK CONFIRMATION
BEAM ID        R01 R02 R03 R04 R05
QUEUE POSIT      1   2   3   4   5
-------------------------------------------------------
S-EVENT - TRACK TRANSITION
BEAM ID        S01
QUEUE POSIT      1
-------------------------------------------------------
T-EVENT - ASSUMED FRIENDLY TRACK
BEAM ID        T01
QUEUE POSIT      1
-------------------------------------------------------
U-EVENT - CONFIRMED FRIENDLY TRACK
BEAM ID        U01 U02
QUEUE POSIT      1   2
-------------------------------------------------------
W-EVENT - SPECIAL ABOVE HORIZON SEARCH
BEAM ID        W01 W02 W03 W04 W05 W06
QUEUE POSIT      1   2   3   4   5   6
-------------------------------------------------------
X-EVENT - SIMULATION DWELL
              NO REQUESTS THIS INTERVAL
-------------------------------------------------------
Y-EVENT - DIAGNOSTIC DWELL
              NO REQUESTS THIS INTERVAL
-------------------------------------------------------
Z-EVENT - DUMMY DWELL
              NO REQUESTS THIS INTERVAL
-------------------------------------------------------

SCHEDULED DWELLS FOR SCHEDULER INTERVAL:    10
-------------------------------------------------------
BEAM ID        G01 G02 G03 G04 G05 I01 I02 I03 I04 I05
RESOURCES       93  86  79  72  65  62  59  56  53  50
DWELL #          1   2   3   4   5   6   7   8   9  10
-------------------------------------------------------
BEAM ID        I06 I07 I08 I09 I10 I11 F01 F02 F03 F04
RESOURCES       47  44  41  38  35  32  25  18  11   4
DWELL #         11  12  13  14  15  16  17  18  19  20
-------------------------------------------------------
BEAM ID        V01
RESOURCES        1
DWELL #         21
-------------------------------------------------------
-------------------------------------------------------
```

```
REQUESTED BEAMS FOR SCHEDULER INTERVAL:    20
-------------------------------------------------------
A-EVENT - ECM BURNTHROUGH
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
B-EVENT - TARGET DEFINITION
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
C-EVENT - SPECIAL TEST
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
F-EVENT - PRE-ENGAGED HOSTILE TARGET
BEAM ID        F01 F02 F03 F04 F05
QUEUE POSIT      1   2   3   4   5
-------------------------------------------------------
E-EVENT - OWN SM-2 MISSILE GUIDANCE
BEAM ID        E01 E02
QUEUE POSIT      1   2
-------------------------------------------------------
I-EVENT - HORIZON SEARCH
BEAM ID        I01 I02 I03 I04 I05 I06 I07 I08 I09 I10
QUEUE POSIT      1   2   3   4   5   6   7   8   9  10
-------------------------------------------------------
P-EVENT - UNEVALUATED TRACK
BEAM ID        P01 P02
QUEUE POSIT      1   2
-------------------------------------------------------
Q-EVENT - CONTROLLED FRIENDLY TRACK
BEAM ID        Q01 Q02 Q03 Q04
QUEUE POSIT      1   2   3   4
-------------------------------------------------------
R-EVENT - TRACK CONFIRMATION
BEAM ID        R01 R02 R03 R04 R05
QUEUE POSIT      1   2   3   4   5
-------------------------------------------------------
S-EVENT - TRACK TRANSITION
BEAM ID        S01
QUEUE POSIT      1
-------------------------------------------------------
T-EVENT - ASSUMED FRIENDLY TRACK
BEAM ID        T01
QUEUE POSIT      1
-------------------------------------------------------
U-EVENT - CONFIRMED FRIENDLY TRACK
BEAM ID        U01 U02
QUEUE POSIT      1   2
-------------------------------------------------------
D-EVENT - ENGAGED HOSTILE TARGET
BEAM ID        D01 D02 D03 D04 D05
QUEUE POSIT      1   2   3   4   5
-------------------------------------------------------
O-EVENT - ASSUMED HOSTILE TRACK
BEAM ID        O01 O02 O03 O04 O05
QUEUE POSIT      1   2   3   4   5
-------------------------------------------------------
N-EVENT - CONFIRMED HOSTILE TRACK
BEAM ID        N01 N02 N03 N04
QUEUE POSIT      1   2   3   4
-------------------------------------------------------
H-EVENT - HIGH PRI TRACK CONFIRMATION
BEAM ID        H01 H02
QUEUE POSIT      1   2
-------------------------------------------------------
G-EVENT - HIGH PRI TRACK TRANSITION
BEAM ID        G01 G02 G03 G04 G05
QUEUE POSIT      1   2   3   4   5
-------------------------------------------------------
```

```
V-EVENT - ABOVE HORIZON SEARCH
BEAM ID        V01 V02 V03 V04 V05 V06 V07 V08 V09 V10
QUEUE POSIT      1   2   3   4   5   6   7   8   9  10
-------------------------------------------------------
J-EVENT - SPECIAL ECM BURNTHROUGH
              NO REQUESTS THIS INTERVAL
-------------------------------------------------------
K-EVENT - SPECIAL TARGET DEFINITION
              NO REQUESTS THIS INTERVAL
-------------------------------------------------------
L-EVENT - SPECIAL MANUAL SCAN
              NO REQUESTS THIS INTERVAL
-------------------------------------------------------
M-EVENT - SPECIAL TARGET ACQUISITION
              NO REQUESTS THIS INTERVAL
-------------------------------------------------------
N-EVENT - SPECIAL ABOVE HORIZON SEARCH
BEAM ID        W01 W02 W03 W04 W05 W06
QUEUE POSIT      1   2   3   4   5   6
-------------------------------------------------------
X-EVENT - SIMULATION DWELL
              NO REQUESTS THIS INTERVAL
-------------------------------------------------------
Y-EVENT - DIAGNOSTIC DWELL
              NO REQUESTS THIS INTERVAL
-------------------------------------------------------
Z-EVENT - DUMMY DWELL
              NO REQUESTS THIS INTERVAL
-------------------------------------------------------

SCHEDULED DWELLS FOR SCHEDULER INTERVAL:    20
-------------------------------------------------------
BEAM ID        F01 F02 F03 F04 F05 E01 E02 I01 I02 I03
RESOURCES       93  86  79  72  65  58  51  48  45  42
DWELL #          1   2   3   4   5   6   7   8   9  10
-------------------------------------------------------
BEAM ID        I04 I05 I06 I07 I08 I09 I10 I11 P01 P02
RESOURCES       39  36  33  30  27  24  21  18  11   4
DWELL #         11  12  13  14  15  16  17  18  19  20
-------------------------------------------------------
BEAM ID        V01
RESOURCES        1
DWELL #         21
-------------------------------------------------------
-------------------------------------------------------
```

```
REQUESTED BEAMS FOR SCHEDULER INTERVAL:    30
-----------------------------------------------------
A-EVENT - ECM BURNTHROUGH
             NO REQUESTS THIS INTERVAL
-----------------------------------------------------
B-EVENT - TARGET DEFINITION
             NO REQUESTS THIS INTERVAL
-----------------------------------------------------
C-EVENT - SPECIAL TEST
             NO REQUESTS THIS INTERVAL
-----------------------------------------------------
F-EVENT - PRE-ENGAGED HOSTILE TARGET
BEAM ID       F01 F02 F03 F04 F05
QUEUE POSIT     1   2   3   4   5
-----------------------------------------------------
E-EVENT - OWN SM-2 MISSILE GUIDANCE
BEAM ID       E01 E02
QUEUE POSIT     1   2
-----------------------------------------------------
D-EVENT - ENGAGED HOSTILE TARGET
BEAM ID       D01 D02 D03 D04 D05
QUEUE POSIT     1   2   3   4   5
-----------------------------------------------------
H-EVENT - HIGH PRI TRACK CONFIRMATION
BEAM ID       H01 H02
QUEUE POSIT     1   2
-----------------------------------------------------
U-EVENT - CONFIRMED FRIENDLY TRACK
BEAM ID       U01 U02
QUEUE POSIT     1   2
-----------------------------------------------------
I-EVENT - HORIZON SEARCH
BEAM ID       I01 I02 I03 I04 I05 I06 I07 I08 I09 I10
QUEUE POSIT     1   2   3   4   5   6   7   8   9  10
-----------------------------------------------------
S-EVENT - TRACK TRANSITION
BEAM ID       S01
QUEUE POSIT     1
-----------------------------------------------------
T-EVENT - ASSUMED FRIENDLY TRACK
BEAM ID       T01
QUEUE POSIT     1
-----------------------------------------------------
R-EVENT - TRACK CONFIRMATION
BEAM ID       R01 R02 R03 R04 R05
QUEUE POSIT     1   2   3   4   5
-----------------------------------------------------
G-EVENT - HIGH PRI TRACK TRANSITION
BEAM ID       G01 G02 G03 G04 G05
QUEUE POSIT     1   2   3   4   5
-----------------------------------------------------
Q-EVENT - CONTROLLED FRIENDLY TRACK
BEAM ID       Q01 Q02 Q03 Q04
QUEUE POSIT     1   2   3   4
-----------------------------------------------------
P-EVENT - UNEVALUATED TRACK
BEAM ID       P01 P02
QUEUE POSIT     1   2
-----------------------------------------------------
O-EVENT - ASSUMED HOSTILE TRACK
BEAM ID       O01 O02 O03 O04 O05
QUEUE POSIT     1   2   3   4   5
-----------------------------------------------------
V-EVENT - ABOVE HORIZON SEARCH
BEAM ID       V01 V02 V03 V04 V05 V06 V07 V08 V09 V10
QUEUE POSIT     1   2   3   4   5   6   7   8   9  10
-----------------------------------------------------
```

```
N-EVENT - CONFIRMED HOSTILE TRACK
BEAM ID        N01 N02 N03 N04
QUEUE POSIT      1   2   3   4
-------------------------------------------------------
J-EVENT - SPECIAL ECM BURNTHROUGH
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
K-EVENT - SPECIAL TARGET DEFINITION
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
L-EVENT - SPECIAL MANUAL SCAN
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
M-EVENT - SPECIAL TARGET ACQUISITION
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
W-EVENT - SPECIAL ABOVE HORIZON SEARCH
BEAM ID        W01 W02 W03 W04 W05 W06
QUEUE POSIT      1   2   3   4   5   6
-------------------------------------------------------
X-EVENT - SIMULATION DWELL
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
Y-EVENT - DIAGNOSTIC DWELL
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
Z-EVENT - DUMMY DWELL
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------

SCHEDULED DWELLS FOR SCHEDULER INTERVAL:    30
-------------------------------------------------------
BEAM ID        F01 F02 F03 F04 F05 E01 E02 D01 D02 D03
RESOURCES       93  86  79  72  65  58  51  44  37  30
DWELL #          1   2   3   4   5   6   7   8   9  10
-------------------------------------------------------
BEAM ID        D04 D05 H01 H02
RESOURCES       23  16   9   2
DWELL #         11  12  13  14
-------------------------------------------------------
-------------------------------------------------------
```

```
REQUESTED BEAMS FOR SCHEDULER INTERVAL:    40
-------------------------------------------------------
A-EVENT - ECM BURNTHROUGH
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
B-EVENT - TARGET DEFINITION
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
C-EVENT - SPECIAL TEST
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
E-EVENT - OWN SM-2 MISSILE GUIDANCE
BEAM ID        E01 E02
QUEUE POSIT      1   2
-------------------------------------------------------
I-EVENT - HORIZON SEARCH
BEAM ID        I01 I02 I03 I04 I05 I06 I07 I08 I09 I10
QUEUE POSIT      1   2   3   4   5   6   7   8   9  10
-------------------------------------------------------
D-EVENT - ENGAGED HOSTILE TARGET
BEAM ID        D01 D02 D03 D04 D05
QUEUE POSIT      1   2   3   4   5
-------------------------------------------------------
H-EVENT - HIGH PRI TRACK CONFIRMATION
BEAM ID        H01 H02
QUEUE POSIT      1   2
-------------------------------------------------------
G-EVENT - HIGH PRI TRACK TRANSITION
BEAM ID        G01 G02 G03 G04 G05
QUEUE POSIT      1   2   3   4   5
-------------------------------------------------------
U-EVENT - CONFIRMED FRIENDLY TRACK
BEAM ID        U01 U02
QUEUE POSIT      1   2
-------------------------------------------------------
F-EVENT - PRE-ENGAGED HOSTILE TARGET
BEAM ID        F01 F02 F03 F04 F05
QUEUE POSIT      1   2   3   4   5
-------------------------------------------------------
S-EVENT - TRACK TRANSITION
BEAM ID        S01
QUEUE POSIT      1
-------------------------------------------------------
T-EVENT - ASSUMED FRIENDLY TRACK
BEAM ID        T01
QUEUE POSIT      1
-------------------------------------------------------
R-EVENT - TRACK CONFIRMATION
BEAM ID        R01 R02 R03 R04 R05
QUEUE POSIT      1   2   3   4   5
-------------------------------------------------------
Q-EVENT - CONTROLLED FRIENDLY TRACK
BEAM ID        Q01 Q02 Q03 Q04
QUEUE POSIT      1   2   3   4
-------------------------------------------------------
P-EVENT - UNEVALUATED TRACK
BEAM ID        P01 P02
QUEUE POSIT      1   2
-------------------------------------------------------
O-EVENT - ASSUMED HOSTILE TRACK
BEAM ID        O01 O02 O03 O04 O05
QUEUE POSIT      1   2   3   4   5
-------------------------------------------------------
V-EVENT - ABOVE HORIZON SEARCH
BEAM ID        V01 V02 V03 V04 V05 V06 V07 V08 V09 V10
QUEUE POSIT      1   2   3   4   5   6   7   8   9  10
-------------------------------------------------------
```

```
N-EVENT - CONFIRMED HOSTILE TRACK
BEAM ID        N01 N02 N03 N04
QUEUE POSIT      1   2   3   4
-------------------------------------------------------
J-EVENT - SPECIAL ECM BURNTHROUGH
              NO REQUESTS THIS INTERVAL
-------------------------------------------------------
K-EVENT - SPECIAL TARGET DEFINITION
              NO REQUESTS THIS INTERVAL
-------------------------------------------------------
L-EVENT - SPECIAL MANUAL SCAN
              NO REQUESTS THIS INTERVAL
-------------------------------------------------------
M-EVENT - SPECIAL TARGET ACQUISITION
              NO REQUESTS THIS INTERVAL
-------------------------------------------------------
W-EVENT - SPECIAL ABOVE HORIZON SEARCH
BEAM ID        W01 W02 W03 W04 W05 W06
QUEUE POSIT      1   2   3   4   5   6
-------------------------------------------------------
X-EVENT - SIMULATION DWELL
              NO REQUESTS THIS INTERVAL
-------------------------------------------------------
Y-EVENT - DIAGNOSTIC DWELL
              NO REQUESTS THIS INTERVAL
-------------------------------------------------------
Z-EVENT - DUMMY DWELL
              NO REQUESTS THIS INTERVAL
-------------------------------------------------------

SCHEDULED DWELLS FOR SCHEDULER INTERVAL:    40
-------------------------------------------------------
BEAM ID        E01 E02 I01 I02 I03 I04 I05 I06 I07 I08
RESOURCES       93  86  83  80  77  74  71  68  65  62
DWELL #          1   2   3   4   5   6   7   8   9  10
-------------------------------------------------------
BEAM ID        I09 I10 I11 D01 D02 D03 D04 D05 H01 H02
RESOURCES       59  56  53  46  39  32  25  18  11   4
DWELL #         11  12  13  14  15  16  17  18  19  20
-------------------------------------------------------
BEAM ID        V01
RESOURCES        1
DWELL #         21
-------------------------------------------------------
-------------------------------------------------------
```

```
REQUESTED BEAMS FOR SCHEDULER INTERVAL:    50
-------------------------------------------------------
A-EVENT - ECM BURNTHROUGH
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
B-EVENT - TARGET DEFINITION
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
C-EVENT - SPECIAL TEST
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
D-EVENT - ENGAGED HOSTILE TARGET
BEAM ID        D01 D02 D03 D04 D05
QUEUE POSIT      1   2   3   4   5
-------------------------------------------------------
H-EVENT - HIGH PRI TRACK CONFIRMATION
BEAM ID        H01 H02
QUEUE POSIT      1   2
-------------------------------------------------------
I-EVENT - HORIZON SEARCH
BEAM ID        I01 I02 I03 I04 I05 I06 I07 I08 I09 I10
QUEUE POSIT      1   2   3   4   5   6   7   8   9  10
-------------------------------------------------------
O-EVENT - ASSUMED HOSTILE TRACK
BEAM ID        O01 O02 O03 O04 O05
QUEUE POSIT      1   2   3   4   5
-------------------------------------------------------
G-EVENT - HIGH PRI TRACK TRANSITION
BEAM ID        G01 G02 G03 G04 G05
QUEUE POSIT      1   2   3   4   5
-------------------------------------------------------
P-EVENT - UNEVALUATED TRACK
BEAM ID        P01 P02
QUEUE POSIT      1   2
-------------------------------------------------------
F-EVENT - PRE-ENGAGED HOSTILE TARGET
BEAM ID        F01 F02 F03 F04 F05
QUEUE POSIT      1   2   3   4   5
-------------------------------------------------------
Q-EVENT - CONTROLLED FRIENDLY TRACK
BEAM ID        Q01 Q02 Q03 Q04
QUEUE POSIT      1   2   3   4
-------------------------------------------------------
N-EVENT - CONFIRMED HOSTILE TRACK
BEAM ID        N01 N02 N03 N04
QUEUE POSIT      1   2   3   4
-------------------------------------------------------
E-EVENT - OWN SM-2 MISSILE GUIDANCE
BEAM ID        E01 E02
QUEUE POSIT      1   2
-------------------------------------------------------
U-EVENT - CONFIRMED FRIENDLY TRACK
BEAM ID        U01 U02
QUEUE POSIT      1   2
-------------------------------------------------------
S-EVENT - TRACK TRANSITION
BEAM ID        S01
QUEUE POSIT      1
-------------------------------------------------------
T-EVENT - ASSUMED FRIENDLY TRACK
BEAM ID        T01
QUEUE POSIT      1
-------------------------------------------------------
V-EVENT - ABOVE HORIZON SEARCH
BEAM ID        V01 V02 V03 V04 V05 V06 V07 V08 V09 V10
QUEUE POSIT      1   2   3   4   5   6   7   8   9  10
-------------------------------------------------------
```

```
R-EVENT - TRACK CONFIRMATION
BEAM ID       R01 R02 R03 R04 R05
QUEUE POSIT     1   2   3   4   5
-------------------------------------------------------
J-EVENT - SPECIAL ECM BURNTHROUGH
            NO REQUESTS THIS INTERVAL
-------------------------------------------------------
K-EVENT - SPECIAL TARGET DEFINITION
            NO REQUESTS THIS INTERVAL
-------------------------------------------------------
L-EVENT - SPECIAL MANUAL SCAN
            NO REQUESTS THIS INTERVAL
-------------------------------------------------------
M-EVENT - SPECIAL TARGET ACQUISITION
            NO REQUESTS THIS INTERVAL
-------------------------------------------------------
W-EVENT - SPECIAL ABOVE HORIZON SEARCH
BEAM ID       W01 W02 W03 W04 W05 W06
QUEUE POSIT     1   2   3   4   5   6
-------------------------------------------------------
X-EVENT - SIMULATION DWELL
            NO REQUESTS THIS INTERVAL
-------------------------------------------------------
Y-EVENT - DIAGNOSTIC DWELL
            NO REQUESTS THIS INTERVAL
-------------------------------------------------------
Z-EVENT - DUMMY DWELL
            NO REQUESTS THIS INTERVAL
-------------------------------------------------------

SCHEDULED DWELLS FOR SCHEDULER INTERVAL:    50
-------------------------------------------------------
BEAM ID       D01 D02 D03 D04 D05 H01 H02 I01 I02 I03
RESOURCES      83  86  79  72  65  58  51  48  45  42
DWELL #         1   2   3   4   5   6   7   8   9  10
-------------------------------------------------------
BEAM ID       I04 I05 I06 I07 I08 I09 I10 I11 O01 O02
RESOURCES      39  36  33  30  27  24  21  18  11   4
DWELL #        11  12  13  14  15  16  17  18  19  20
-------------------------------------------------------
BEAM ID       V01
RESOURCES       1
DWELL #        21
-------------------------------------------------------
-------------------------------------------------------
```

```
REQUESTED BEAMS FOR SCHEDULER INTERVAL:    60
-----------------------------------------------------
A-EVENT - ECM BURNTHROUGH
             NO REQUESTS THIS INTERVAL
-----------------------------------------------------
B-EVENT - TARGET DEFINITION
             NO REQUESTS THIS INTERVAL
-----------------------------------------------------
C-EVENT - SPECIAL TEST
             NO REQUESTS THIS INTERVAL
-----------------------------------------------------
H-EVENT - HIGH PRI TRACK CONFIRMATION
BEAM ID        H01 H02
QUEUE POSIT      1   2
-----------------------------------------------------
G-EVENT - HIGH PRI TRACK TRANSITION
BEAM ID        G01 G02 G03 G04 G05
QUEUE POSIT      1   2   3   4   5
-----------------------------------------------------
I-EVENT - HORIZON SEARCH
BEAM ID        I01 I02 I03 I04 I05 I06 I07 I08 I09 I10
QUEUE POSIT      1   2   3   4   5   6   7   8   9  10
-----------------------------------------------------
Q-EVENT - CONTROLLED FRIENDLY TRACK
BEAM ID        Q01 Q02 Q03 Q04
QUEUE POSIT      1   2   3   4
-----------------------------------------------------
R-EVENT - TRACK CONFIRMATION
BEAM ID        R01 R02 R03 R04 R05
QUEUE POSIT      1   2   3   4   5
-----------------------------------------------------
S-EVENT - TRACK TRANSITION
BEAM ID        S01
QUEUE POSIT      1
-----------------------------------------------------
T-EVENT - ASSUMED FRIENDLY TRACK
BEAM ID        T01
QUEUE POSIT      1
-----------------------------------------------------
F-EVENT - PRE-ENGAGED HOSTILE TARGET
BEAM ID        F01 F02 F03 F04 F05
QUEUE POSIT      1   2   3   4   5
-----------------------------------------------------
E-EVENT - OWN SM-2 MISSILE GUIDANCE
BEAM ID        E01 E02
QUEUE POSIT      1   2
-----------------------------------------------------
D-EVENT - ENGAGED HOSTILE TARGET
BEAM ID        D01 D02 D03 D04 D05
QUEUE POSIT      1   2   3   4   5
-----------------------------------------------------
P-EVENT - UNEVALUATED TRACK
BEAM ID        P01 P02
QUEUE POSIT      1   2
-----------------------------------------------------
O-EVENT - ASSUMED HOSTILE TRACK
BEAM ID        O01 O02 O03 O04 O05
QUEUE POSIT      1   2   3   4   5
-----------------------------------------------------
N-EVENT - CONFIRMED HOSTILE TRACK
BEAM ID        N01 N02 N03 N04
QUEUE POSIT      1   2   3   4
-----------------------------------------------------
V-EVENT - ABOVE HORIZON SEARCH
BEAM ID        V01 V02 V03 V04 V05 V06 V07 V08 V09 V10
QUEUE POSIT      1   2   3   4   5   6   7   8   9  10
-----------------------------------------------------
```

```
U-EVENT - CONFIRMED FRIENDLY TRACK
BEAM ID        U01 U02
QUEUE POSIT      1   2
-------------------------------------------------------
J-EVENT - SPECIAL ECM BURNTHROUGH
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
K-EVENT - SPECIAL TARGET DEFINITION
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
L-EVENT - SPECIAL MANUAL SCAN
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
M-EVENT - SPECIAL TARGET ACQUISITION
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
W-EVENT - SPECIAL ABOVE HORIZON SEARCH
BEAM ID        W01 W02 W03 W04 W05 W06
QUEUE POSIT      1   2   3   4   5   6
-------------------------------------------------------
X-EVENT - SIMULATION DWELL
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
Y-EVENT - DIAGNOSTIC DWELL
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
Z-EVENT - DUMMY DWELL
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------

SCHEDULED DWELLS FOR SCHEDULER INTERVAL:    60
-------------------------------------------------------
BEAM ID        H01 H02 G01 G02 G03 G04 G05 I01 I02 I03
RESOURCES       93  86  79  72  65  58  51  48  45  42
DWELL #          1   2   3   4   5   6   7   8   9  10
-------------------------------------------------------
BEAM ID        I04 I05 I06 I07 I08 I09 I10 I11 Q01 Q02
RESOURCES       39  36  33  30  27  24  21  18  11   4
DWELL #         11  12  13  14  15  16  17  18  19  20
-------------------------------------------------------
BEAM ID        V01
RESOURCES        1
DWELL #         21
-------------------------------------------------------
-------------------------------------------------------
```

```
REQUESTED BEAMS FOR SCHEDULER INTERVAL:    70
-----------------------------------------------------
A-EVENT - ECM BURNTHROUGH
             NO REQUESTS THIS INTERVAL
-----------------------------------------------------
B-EVENT - TARGET DEFINITION
             NO REQUESTS THIS INTERVAL
-----------------------------------------------------
C-EVENT - SPECIAL TEST
             NO REQUESTS THIS INTERVAL
-----------------------------------------------------
D-EVENT - ENGAGED HOSTILE TARGET
BEAM ID        D01 D02 D03 D04 D05
QUEUE POSIT      1   2   3   4   5
-----------------------------------------------------
H-EVENT - HIGH PRI TRACK CONFIRMATION
BEAM ID        H01 H02
QUEUE POSIT      1   2
-----------------------------------------------------
G-EVENT - HIGH PRI TRACK TRANSITION
BEAM ID        G01 G02 G03 G04 G05
QUEUE POSIT      1   2   3   4   5
-----------------------------------------------------
F-EVENT - PRE-ENGAGED HOSTILE TARGET
BEAM ID        F01 F02 F03 F04 F05
QUEUE POSIT      1   2   3   4   5
-----------------------------------------------------
E-EVENT - OWN SM-2 MISSILE GUIDANCE
BEAM ID        E01 E02
QUEUE POSIT      1   2
-----------------------------------------------------
I-EVENT - HORIZON SEARCH
BEAM ID        I01 I02 I03 I04 I05 I06 I07 I08 I09 I10
QUEUE POSIT      1   2   3   4   5   6   7   8   9  10
-----------------------------------------------------
S-EVENT - TRACK TRANSITION
BEAM ID        S01
QUEUE POSIT      1
-----------------------------------------------------
U-EVENT - CONFIRMED FRIENDLY TRACK
BEAM ID        U01 U02
QUEUE POSIT      1   2
-----------------------------------------------------
T-EVENT - ASSUMED FRIENDLY TRACK
BEAM ID        T01
QUEUE POSIT      1
-----------------------------------------------------
R-EVENT - TRACK CONFIRMATION
BEAM ID        R01 R02 R03 R04 R05
QUEUE POSIT      1   2   3   4   5
-----------------------------------------------------
Q-EVENT - CONTROLLED FRIENDLY TRACK
BEAM ID        Q01 Q02 Q03 Q04
QUEUE POSIT      1   2   3   4
-----------------------------------------------------
P-EVENT - UNEVALUATED TRACK
BEAM ID        P01 P02
QUEUE POSIT      1   2
-----------------------------------------------------
O-EVENT - ASSUMED HOSTILE TRACK
BEAM ID        O01 O02 O03 O04 O05
QUEUE POSIT      1   2   3   4   5
-----------------------------------------------------
N-EVENT - CONFIRMED HOSTILE TRACK
BEAM ID        N01 N02 N03 N04
QUEUE POSIT      1   2   3   4
-----------------------------------------------------
```

```
V-EVENT - ABOVE HORIZON SEARCH
BEAM ID       V01 V02 V03 V04 V05 V06 V07 V08 V09 V10
QUEUE POSIT     1   2   3   4   5   6   7   8   9  10
-------------------------------------------------------
J-EVENT - SPECIAL ECM BURNTHROUGH
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
K-EVENT - SPECIAL TARGET DEFINITION
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
L-EVENT - SPECIAL MANUAL SCAN
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
M-EVENT - SPECIAL TARGET ACQUISITION
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
W-EVENT - SPECIAL ABOVE HORIZON SEARCH
BEAM ID       W01 W02 W03 W04 W05 W06
QUEUE POSIT     1   2   3   4   5   6
-------------------------------------------------------
X-EVENT - SIMULATION DWELL
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
Y-EVENT - DIAGNOSTIC DWELL
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
Z-EVENT - DUMMY DWELL
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------

SCHEDULED DWELLS FOR SCHEDULER INTERVAL:    70
-------------------------------------------------------
BEAM ID       D01 D02 D03 D04 D05 H01 H02 G01 G02 G03
RESOURCES      93  86  79  72  65  58  51  44  37  30
DWELL #         1   2   3   4   5   6   7   8   9  10
-------------------------------------------------------
BEAM ID       G04 G05 F01 F02
RESOURCES      23  16   9   2
DWELL #        11  12  13  14
-------------------------------------------------------
-------------------------------------------------------
```

```
REQUESTED BEAMS FOR SCHEDULER INTERVAL:    80
-----------------------------------------------------
A-EVENT - ECM BURNTHROUGH
             NO REQUESTS THIS INTERVAL
-----------------------------------------------------
B-EVENT - TARGET DEFINITION
             NO REQUESTS THIS INTERVAL
-----------------------------------------------------
C-EVENT - SPECIAL TEST
             NO REQUESTS THIS INTERVAL
-----------------------------------------------------
G-EVENT - HIGH PRI TRACK TRANSITION
BEAM ID       G01 G02 G03 G04 G05
QUEUE POSIT     1   2   3   4   5
-----------------------------------------------------
F-EVENT - PRE-ENGAGED HOSTILE TARGET
BEAM ID       F01 F02 F03 F04 F05
QUEUE POSIT     1   2   3   4   5
-----------------------------------------------------
E-EVENT - OWN SM-2 MISSILE GUIDANCE
BEAM ID       E01 E02
QUEUE POSIT     1   2
-----------------------------------------------------
D-EVENT - ENGAGED HOSTILE TARGET
BEAM ID       D01 D02 D03 D04 D05
QUEUE POSIT     1   2   3   4   5
-----------------------------------------------------
N-EVENT - CONFIRMED HOSTILE TRACK
BEAM ID       N01 N02 N03 N04
QUEUE POSIT     1   2   3   4
-----------------------------------------------------
I-EVENT - HORIZON SEARCH
BEAM ID       I01 I02 I03 I04 I05 I06 I07 I08 I09 I10
QUEUE POSIT     1   2   3   4   5   6   7   8   9  10
-----------------------------------------------------
O-EVENT - ASSUMED HOSTILE TRACK
BEAM ID       O01 O02 O03 O04 O05
QUEUE POSIT     1   2   3   4   5
-----------------------------------------------------
H-EVENT - HIGH PRI TRACK CONFIRMATION
BEAM ID       H01 H02
QUEUE POSIT     1   2
-----------------------------------------------------
U-EVENT - CONFIRMED FRIENDLY TRACK
BEAM ID       U01 U02
QUEUE POSIT     1   2
-----------------------------------------------------
P-EVENT - UNEVALUATED TRACK
BEAM ID       P01 P02
QUEUE POSIT     1   2
-----------------------------------------------------
S-EVENT - TRACK TRANSITION
BEAM ID       S01
QUEUE POSIT     1
-----------------------------------------------------
T-EVENT - ASSUMED FRIENDLY TRACK
BEAM ID       T01
QUEUE POSIT     1
-----------------------------------------------------
V-EVENT - ABOVE HORIZON SEARCH
BEAM ID       V01 V02 V03 V04 V05 V06 V07 V08 V09 V10
QUEUE POSIT     1   2   3   4   5   6   7   8   9  10
-----------------------------------------------------
R-EVENT - TRACK CONFIRMATION
BEAM ID       R01 R02 R03 R04 R05
QUEUE POSIT     1   2   3   4   5
-----------------------------------------------------
```

```
Q-EVENT - CONTROLLED FRIENDLY TRACK
BEAM ID        Q01 Q02 Q03 Q04
QUEUE POSIT      1   2   3   4
-------------------------------------------------------
J-EVENT - SPECIAL ECM BURNTHROUGH
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
K-EVENT - SPECIAL TARGET DEFINITION
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
L-EVENT - SPECIAL MANUAL SCAN
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
M-EVENT - SPECIAL TARGET ACQUISITION
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
W-EVENT - SPECIAL ABOVE HORIZON SEARCH
BEAM ID        W01 W02 W03 W04 W05 W06
QUEUE POSIT      1   2   3   4   5   6
-------------------------------------------------------
X-EVENT - SIMULATION DWELL
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
Y-EVENT - DIAGNOSTIC DWELL
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
Z-EVENT - DUMMY DWELL
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------

SCHEDULED DWELLS FOR SCHEDULER INTERVAL:    80
-------------------------------------------------------
BEAM ID        G01 G02 G03 G04 G05 F01 F02 F03 F04 F05
RESOURCES       93  36  79  72  65  58  51  44  37  30
DWELL #          1   2   3   4   5   6   7   8   9  10
-------------------------------------------------------
BEAM ID        E01 E02 D01 D02
RESOURCES       23  16   9   2
DWELL #         11  12  13  14
-------------------------------------------------------
-------------------------------------------------------
```

```
REQUESTED BEAMS FOR SCHEDULER INTERVAL:    90
-----------------------------------------------------
A-EVENT - ECM BURNTHROUGH
             NO REQUESTS THIS INTERVAL
-----------------------------------------------------
B-EVENT - TARGET DEFINITION
             NO REQUESTS THIS INTERVAL
-----------------------------------------------------
C-EVENT - SPECIAL TEST
             NO REQUESTS THIS INTERVAL
-----------------------------------------------------
F-EVENT - PRE-ENGAGED HOSTILE TARGET
BEAM ID        F01 F02 F03 F04 F05
QUEUE POSIT      1   2   3   4   5
-----------------------------------------------------
E-EVENT - OWN SM-2 MISSILE GUIDANCE
BEAM ID        E01 E02
QUEUE POSIT      1   2
-----------------------------------------------------
D-EVENT - ENGAGED HOSTILE TARGET
BEAM ID        D01 D02 D03 D04 D05
QUEUE POSIT      1   2   3   4   5
-----------------------------------------------------
H-EVENT - HIGH PRI TRACK CONFIRMATION
BEAM ID        H01 H02
QUEUE POSIT      1   2
-----------------------------------------------------
P-EVENT - UNEVALUATED TRACK
BEAM ID        P01 P02
QUEUE POSIT      1   2
-----------------------------------------------------
I-EVENT - HORIZON SEARCH
BEAM ID        I01 I02 I03 I04 I05 I06 I07 I08 I09 I10
QUEUE POSIT      1   2   3   4   5   6   7   8   9  10
-----------------------------------------------------
G-EVENT - HIGH PRI TRACK TRANSITION
BEAM ID        G01 G02 G03 G04 G05
QUEUE POSIT      1   2   3   4   5
-----------------------------------------------------
O-EVENT - ASSUMED HOSTILE TRACK
BEAM ID        O01 O02 O03 O04 O05
QUEUE POSIT      1   2   3   4   5
-----------------------------------------------------
N-EVENT - CONFIRMED HOSTILE TRACK
BEAM ID        N01 N02 N03 N04
QUEUE POSIT      1   2   3   4
-----------------------------------------------------
Q-EVENT - CONTROLLED FRIENDLY TRACK
BEAM ID        Q01 Q02 Q03 Q04
QUEUE POSIT      1   2   3   4
-----------------------------------------------------
U-EVENT - CONFIRMED FRIENDLY TRACK
BEAM ID        U01 U02
QUEUE POSIT      1   2
-----------------------------------------------------
R-EVENT - TRACK CONFIRMATION
BEAM ID        R01 R02 R03 R04 R05
QUEUE POSIT      1   2   3   4   5
-----------------------------------------------------
S-EVENT - TRACK TRANSITION
BEAM ID        S01
QUEUE POSIT      1
-----------------------------------------------------
V-EVENT - ABOVE HORIZON SEARCH
BEAM ID        V01 V02 V03 V04 V05 V06 V07 V08 V09 V10
QUEUE POSIT      1   2   3   4   5   6   7   8   9  10
-----------------------------------------------------
```

```
T-EVENT - ASSUMED FRIENDLY TRACK
BEAM ID        T01
QUEUE POSIT      1
-------------------------------------------------------
J-EVENT - SPECIAL ECM BURNTHROUGH
              NO REQUESTS THIS INTERVAL
-------------------------------------------------------
K-EVENT - SPECIAL TARGET DEFINITION
              NO REQUESTS THIS INTERVAL
-------------------------------------------------------
L-EVENT - SPECIAL MANUAL SCAN
              NO REQUESTS THIS INTERVAL
-------------------------------------------------------
M-EVENT - SPECIAL TARGET ACQUISITION
              NO REQUESTS THIS INTERVAL
-------------------------------------------------------
W-EVENT - SPECIAL ABOVE HORIZON SEARCH
BEAM ID        W01 W02 W03 W04 W05 W06
QUEUE POSIT      1   2   3   4   5   6
-------------------------------------------------------
X-EVENT - SIMULATION DWELL
              NO REQUESTS THIS INTERVAL
-------------------------------------------------------
Y-EVENT - DIAGNOSTIC DWELL
              NO REQUESTS THIS INTERVAL
-------------------------------------------------------
Z-EVENT - DUMMY DWELL
              NO REQUESTS THIS INTERVAL
-------------------------------------------------------

SCHEDULED DWELLS FOR SCHEDULER INTERVAL:    90
-------------------------------------------------------
BEAM ID        F01 F02 F03 F04 F05 E01 E02 D01 D02 D03
RESOURCES       93  86  79  72  65  58  51  44  37  30
DWELL #          1   2   3   4   5   6   7   8   9  10
-------------------------------------------------------
BEAM ID        D04 D05 H01 H02
RESOURCES       23  16   9   2
DWELL #         11  12  13  14
-------------------------------------------------------
-------------------------------------------------------
```

```
REQUESTED BEAMS FOR SCHEDULER INTERVAL:   100
-----------------------------------------------------
A-EVENT - ECM BURNTHROUGH
             NO REQUESTS THIS INTERVAL
-----------------------------------------------------
B-EVENT - TARGET DEFINITION
             NO REQUESTS THIS INTERVAL
-----------------------------------------------------
C-EVENT - SPECIAL TEST
             NO REQUESTS THIS INTERVAL
-----------------------------------------------------
H-EVENT - HIGH PRI TRACK CONFIRMATION
BEAM ID       H01 H02
QUEUE POSIT     1   2
-----------------------------------------------------
I-EVENT - HORIZON SEARCH
BEAM ID       I01 I02 I03 I04 I05 I06 I07 I08 I09 I10
QUEUE POSIT     1   2   3   4   5   6   7   8   9  10
-----------------------------------------------------
E-EVENT - OWN SM-2 MISSILE GUIDANCE
BEAM ID       E01 E02
QUEUE POSIT     1   2
-----------------------------------------------------
Q-EVENT - CONTROLLED FRIENDLY TRACK
BEAM ID       Q01 Q02 Q03 Q04
QUEUE POSIT     1   2   3   4
-----------------------------------------------------
S-EVENT - TRACK TRANSITION
BEAM ID       S01
QUEUE POSIT     1
-----------------------------------------------------
R-EVENT - TRACK CONFIRMATION
BEAM ID       R01 R02 R03 R04 R05
QUEUE POSIT     1   2   3   4   5
-----------------------------------------------------
D-EVENT - ENGAGED HOSTILE TARGET
BEAM ID       D01 D02 D03 D04 D05
QUEUE POSIT     1   2   3   4   5
-----------------------------------------------------
G-EVENT - HIGH PRI TRACK TRANSITION
BEAM ID       G01 G02 G03 G04 G05
QUEUE POSIT     1   2   3   4   5
-----------------------------------------------------
P-EVENT - UNEVALUATED TRACK
BEAM ID       P01 P02
QUEUE POSIT     1   2
-----------------------------------------------------
F-EVENT - PRE-ENGAGED HOSTILE TARGET
BEAM ID       F01 F02 F03 F04 F05
QUEUE POSIT     1   2   3   4   5
-----------------------------------------------------
O-EVENT - ASSUMED HOSTILE TRACK
BEAM ID       O01 O02 O03 O04 O05
QUEUE POSIT     1   2   3   4   5
-----------------------------------------------------
N-EVENT - CONFIRMED HOSTILE TRACK
BEAM ID       N01 N02 N03 N04
QUEUE POSIT     1   2   3   4
-----------------------------------------------------
U-EVENT - CONFIRMED FRIENDLY TRACK
BEAM ID       U01 U02
QUEUE POSIT     1   2
-----------------------------------------------------
T-EVENT - ASSUMED FRIENDLY TRACK
BEAM ID       T01
QUEUE POSIT     1
-----------------------------------------------------
```

```
V-EVENT - ABOVE HORIZON SEARCH
BEAM ID       V01 V02 V03 V04 V05 V06 V07 V08 V09 V10
QUEUE POSIT     1   2   3   4   5   6   7   8   9  10
-------------------------------------------------------
J-EVENT - SPECIAL ECM BURNTHROUGH
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
K-EVENT - SPECIAL TARGET DEFINITION
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
L-EVENT - SPECIAL MANUAL SCAN
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
M-EVENT - SPECIAL TARGET ACQUISITION
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
W-EVENT - SPECIAL ABOVE HORIZON SEARCH
BEAM ID       W01 W02 W03 W04 W05 W06
QUEUE POSIT     1   2   3   4   5   6
-------------------------------------------------------
X-EVENT - SIMULATION DWELL
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
Y-EVENT - DIAGNOSTIC DWELL
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------
Z-EVENT - DUMMY DWELL
             NO REQUESTS THIS INTERVAL
-------------------------------------------------------

SCHEDULED DWELLS FOR SCHEDULER INTERVAL:   100
-------------------------------------------------------
BEAM ID       H01 H02 I01 I02 I03 I04 I05 I06 I07 I08
RESOURCES      93  86  83  80  77  74  71  68  65  62
DWELL #         1   2   3   4   5   6   7   8   9  10
-------------------------------------------------------
BEAM ID       I09 I10 I11 E01 E02 Q01 Q02 Q03 Q04 S01
RESOURCES      59  56  53  46  39  32  25  18  11   4
DWELL #        11  12  13  14  15  16  17  18  19  20
-------------------------------------------------------
BEAM ID       V01
RESOURCES       1
DWELL #        21
-------------------------------------------------------
-------------------------------------------------------
```

LIST OF REFERENCES

1. Gayler, R., *A Multi-microprocessing Approach to The AEGIS Combat System*, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1980.

2. Dilmore, W. D., *The INTEL MCS-86 as a Functionally Dedicated Microprocessor in AN/SPY-1A Radar Control*, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1980.

3. Riche, R. S. and Williams, C. E., *A Software Foundation For AN/SPY-1A Radar Control*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1981.

4. U.S. Department of Defense, Specification ANSI/MIL-STD-1815A, *Military Standard Ada Programming Language*, 22 January 1983.

5. Grant, P.M., *A Multi-Microprocessor Based Model of The AEGIS AN/SPY-1A Radar Control: Raaar Scheauler*, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1982

6. RR Software, Inc., *Janus/Ada Package Users Manuals*, 8086 version 3.2, March 1983.

BIBLIOGRAPHY

Barnes, J. G. P., *Programing in Ada*, 2d ed., Addison-Wesley Publishers Limited, 1984.

Fairley, R. E., *Software Engineering Concepts*, McGraw-Hill, Inc., 1985.

Johnson P. I., *The Ada Primer*, McGraw-Hill, Inc., 1985.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943	1
4.	Dr. Uno R. Kodres, Code 52Kr Department of Computer Science Naval Postgraduate School Monterey, California 93943	3
5.	CDR Gary S. Baker, Code 52Bj Department of Computer Science Naval Postgraduate School Monterey, California 93943	1
6.	Daniel Green, Code 20F Naval Surface Weapons Center Dahlgren, Virginia 22449	1
7.	CAPT. J.Hood, USN PMS 400B5 Naval Sea Systems Command Washington, D.C. 20362	1
8.	RCA AEGIS Repository RCA Corporation Government Systems Division Mail Shop 127-327 Moorestown, New Jersey 08057	1
9.	Library (Code E33-05) Naval Surface Weapons Center Dahlgren, Virginia 22449	1
10.	Dr. M. J. Gralia Applied Physics Laboratory John Hopkins Road Laurel, Maryland 20707	1
11.	Dana Small, Code 8242 Naval Ocean Systems Center San Diego, California 92152	1
12.	LT J. H. Purdum Long Beach Naval Shipyard Long Beach, California 90822-5099	2